The secrets behind Guilty Gear Xrd -SIGN-'s real-time 3D Anime graphics

4Gamer.net

This new series, ***"Examining Game Graphics"*** focuses on the graphics of specific titles, and explains the systems and technology behind them.
I have been developing ***"Nishikawa Zenji's 3D Game Ecstasy"*** as a series, but since the scope of coverage has become too broad, I would like to deal with technical explanations of specific titles in this new series.
This 1st commemorative issue will feature **GUILTY GEAR Xrd -SIGN-**, a fighting game developed by Arc System Works and rocking arcades since 2014/2.

Original Article 1 / 2 .: Writer_西川善司/Zenji Nishikawa ;
Photographer_佐々木秀二/Shuji Sasaki , 2014

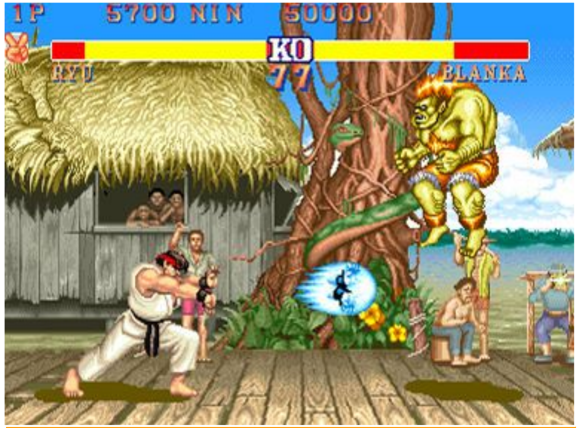
Translation and slides by .: @LeoBGKK on Twitter
Original rough translation by Chev on Polycount Boards

Guilty Gear in Full 3D Graphics

Guilty Gear in full 3D Graphics

There are various theories on the origins of fighting games, but Capcom's Street Fighter series is probably the de facto **originator of the real-time, face-to-face, side scrolling system, where inputting lever directions plus button presses controls the system**.

Eventually, the 2D fighting game turned to 3D graphics in the early 1990s and came to be called "3D fighting game". Virtua Fighter being the original 3D Fighting Game. What's interesting is that **not all fighting games transitioned to 3D graphics** after that. As a result, the traditional "drawn" and "pixel" games were given a new name, "2D fighting games," and survived.



Street Fighter II © CAPCOM U.S.A

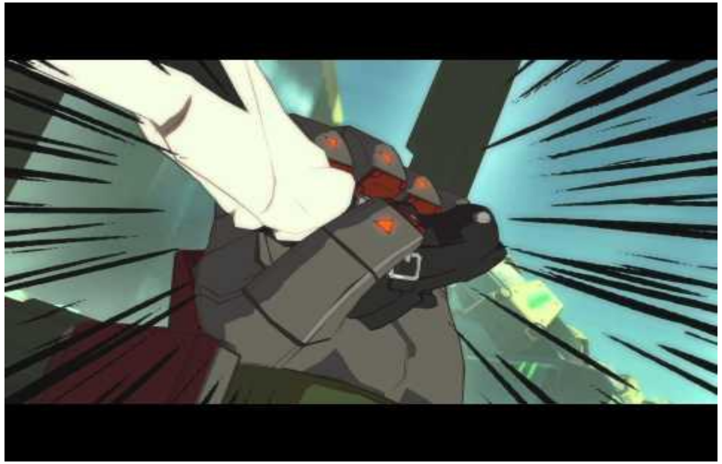
While 3D fighting games have evolved in line with the evolution of GPUs, with more realistic graphics, better animations, and flashier effects, the evolution of 2D fighting games has been more subdued. It's no wonder, as 2D fighting game graphics are ultimately a process of **hand-drawing character animation frames one-by-one**, so the benefits of GPU evolution are not significant.

That's not to say that there haven't been attempts to revolutionize the graphics of 2D fighting games, such as the Capcom vs. SNK series, which uses 2D graphics for characters and dynamic 3D graphics for backgrounds. In recent years, there have also been cases such as The King of Fighters series, where the poses of the characters are generated in 3D graphics and then traced by the drawing staff. Incidentally, this rotoscoping technique is commonly used in recent TV animations.

In recent years, Capcom has created 3D graphics for Street Fighter IV that don't break the pixel atmosphere of Street Fighter II, a big hit in the early 90s. A new style of *"3D graphics but 2D game system"* was born.

Guilty Gear in full 3D Graphics

2D fighting games have been exploring new techniques of expression by adopting 3D graphics technology in one form or another, **but what about GUILTY GEAR Xrd -SIGN-** (GGXrd)? After knowing that this is a 2D fighting game, I'd like you to take a look at the movie below. It's a digest of the gameplay scenes edited by Arc System Works, in the 1920x1080 resolution that will be available on the PS4 version. Check it out.



It may look hand-drawn, but in reality, **it's 100% in 3D.**

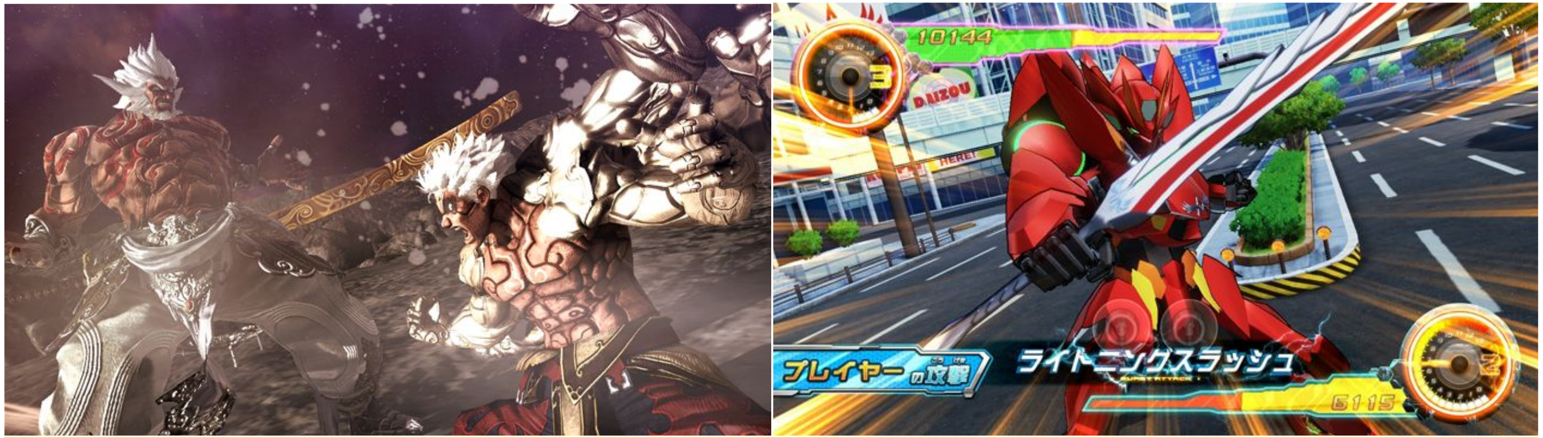
In moments like super special moves, the camera moves boldly from the side view of normal battles to show the dynamic actions of the characters, which is truly a benefit of 3D graphics.

It’s Unreal Engine 3

It’s Unreal Engine 3

GGXrd is based on **Unreal Engine 3** (UE3), a game engine developed by Epic Games.

UE3 is well known as an engine that excels in realistic 3D game graphics, as seen Gears of War and Mass Effect. Of course, game engines are basically designed to be able to handle a wide variety of game production, so from a technical standpoint, it’s reasonable to say that it's possible to achieve anime-style visuals with UE3. However, from a fighting gamer's point of view, the fact that Arc System Works, a company that excels at hand-drawn anime style visuals, has created a new Guilty Gear game using UE3, which excels at so-called "Western" visuals, is hugely impactful.



UE3 has been increasingly adopted by major game studios in Japan. Above, CyberConnect2's "ASURA'S WRATH" , and Square Enix's "Chousoku Henkei Gyrozetter"
ASURA'S WRATH © CAPCOM
Chousoku Henkei Gyrozetter © SQUARE ENIX

Guilty Gear in full 3D Graphics

Why UE3? We first asked the development staff about this, and got the following answers from Daisuke Ishiwatari, the general director, and Takeshi Yamanaka, the director.

Daisuke Ishiwatari— *If I had to give one obvious reason, it would be that we didn't have much time to spare (laughs).*

I'll talk about how we came to the decision to use 3D graphics later, but in order to complete the game using 3D graphics within the allotted production time, we needed to use an existing game engine.

We had previously developed "Guilty Gear 2: Overture," a real-time strategy-style game with 3D graphics, using a from scratch in-house game engine. However, we didn't have the time to improve on this engine and start production.

Daisuke Ishiwatari (General Director) has been involved in the production since the first title in the series, "Guilty Gear" for the original PlayStation, and is the general director of the series. In addition to directing the entire project, he is also in charge of visual production and music composition, making him Mr. Guilty Gear.

Guilty Gear in full 3D Graphics

Takeshi Yamanaka— *Epic Games has announced Unreal Engine 4 (UE4) as their next generation game engine. We started looking for an engine right around that time, and with the introduction of UE4, UE3 was discounted, making it affordable for small and medium studios like ours. This was one of the major factors that led to our adoption.*

Also, Epic Games, who had seen the prototype version of GGXrd, gave it high marks for "bringing out the new potential of UE3," and we were able to receive multifaceted support from them.

GGXrd was an arcade title. Didn't they consider UE4? Or what about other engines, domestic or foreign? Takuro Ieyumi, the lead programmer, answered this point as follows. →

Takeshi Yamanaka (Director). He supervised the storyline, wrote the script, and created the sound effects for the game. He has also been involved in the creation of the worldview for the Guilty Gear series with Ishiwatari. In the past, he's also been in charge of the direction of the BlazBlue series.

Guilty Gear in full 3D Graphics

Takuro Ieyumi— *Of course we considered it. But for GGXrd,* ***we knew that we would be adding and modifying functions to the existing game engine****, so we needed to access to the source code and have a functionally mature engine.*
UE3 was the one that met these requirements. For an engineer like myself, it had a lot of appeal.

It's an engine that has been in use since around 2005, and the feedback from developers who have worked on it before us has made it functionally very stable. We don't want the whole system to become unstable when adding our own functions.

Also, if such a situation were to occur, it would be easier to identify the cause if the engine side was stable. That's why it was also important.

Hidehiko Sakamura, art director and chief animator, and Junya Motomura, lead modeler and technical artist, gave the following evaluation of UE3 from the artist's perspective. →

Takuro Ieyumi (Lead Programmer), who was in charge of overall program development for GGXrd. His previous works include "Battle Fantasia".

Guilty Gear in full 3D Graphics

Hidehiko Sakamura— *The UE3 UDK (development kit), or the "free version" of UE3, is available to the public, and with it, artists like myself can create a sample FPS using our 3D models. It was quite easy to use as a tool.*

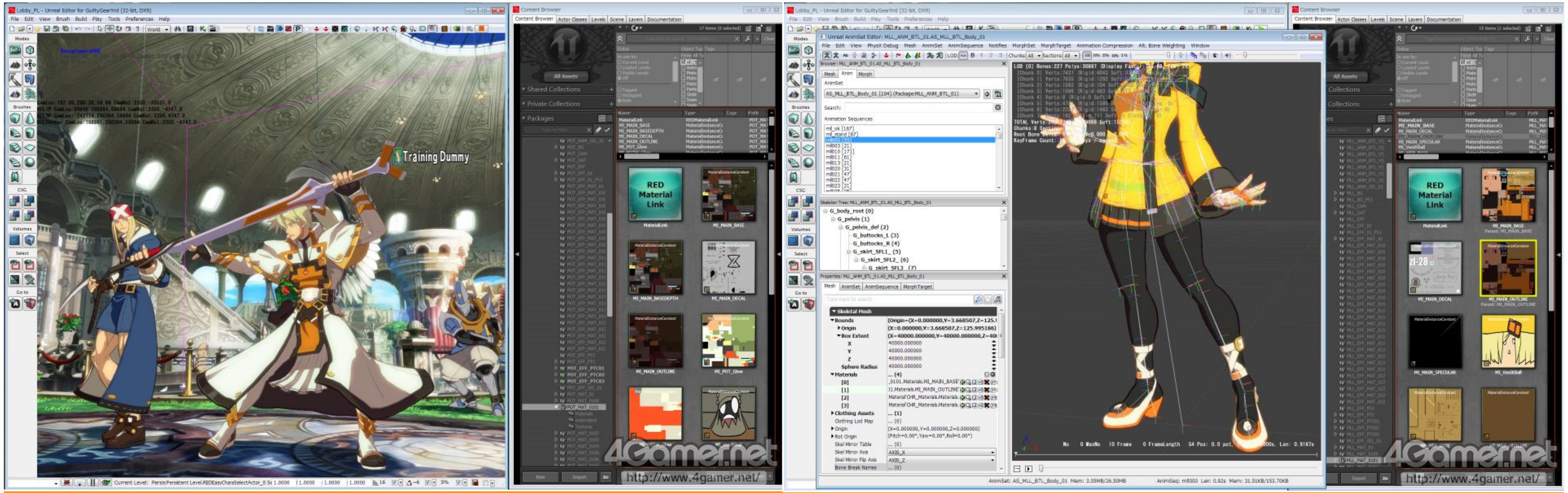
Junya Christopher Motomura— *Our development team doesn't have many programmers, so UE3 was a great tool for us to be able to proceed with development without bothering them. The artists can do a lot of work on their own.*

In a fighting game, various adjustments are made during development, but if the data is exchanged between the artists and programmers each time, development efficiency would suffer. With UE3, the artists were able to set up the game on their own and see what they were creating, so we were able to avoid that problem.

In the GGXrd project, we used the tools provided by UE3 for basic content creation, but for other tasks, particularly the creation of the core 2D fighting game, we used the in-house tools that Arc System Works has used in past projects. Specifically, the in-house tools included a 2D fighting game collision detection tool and a script tool for controlling character actions.

Hidehiko Sakamura (Art Director and Chief Animator) was involved in the character animation design for not only the fighting but also the Story Mode of GGXrd. He was also in charge of animation for Guilty Gear 2 Overture in the past.

It’s Unreal Engine 3

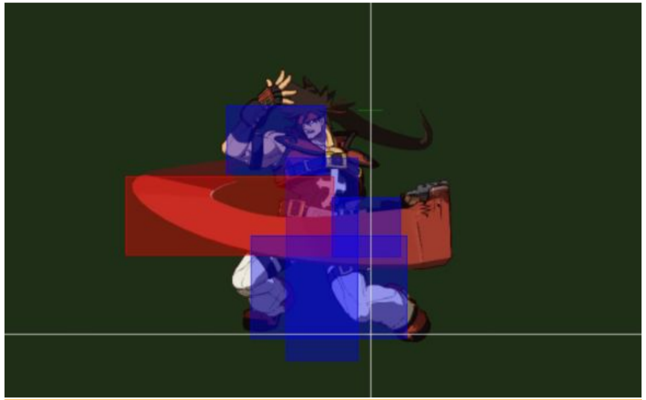


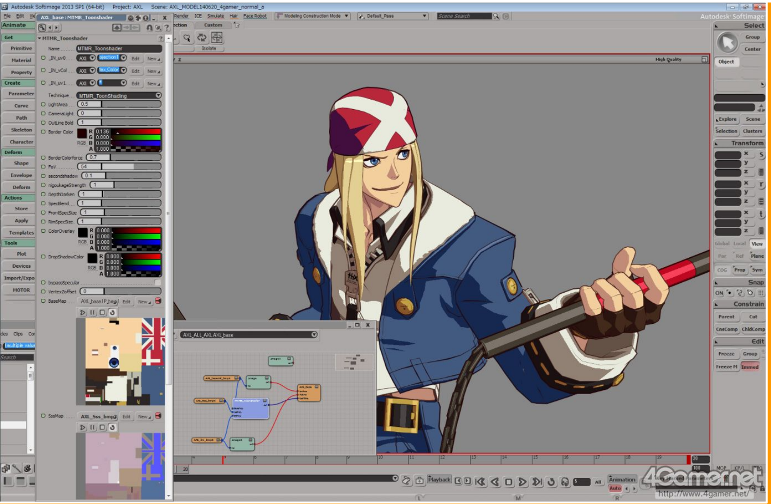
The UE3 Tools

It's Unreal Engine 3

The game logic part of the 2D fighting game **was not created anew**, but was ported to UE3 from the proven tools that had been developed in-house.

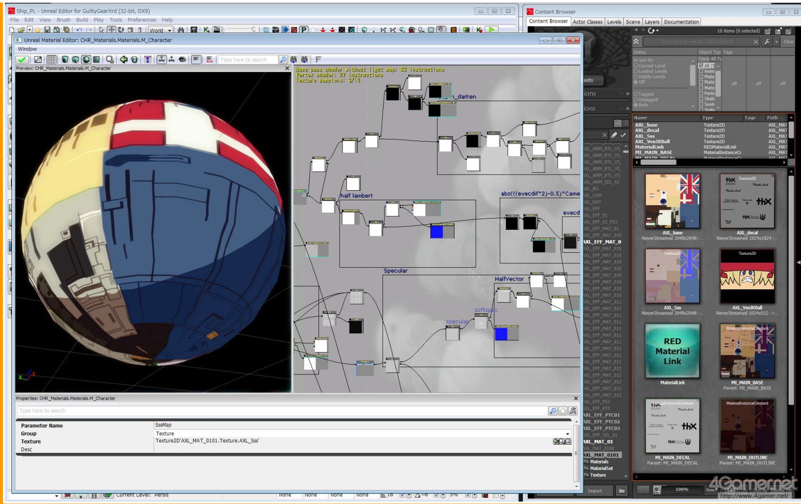
The company's familiar tools not only have a stable track record, but also many people who can use them within the company. Considering the need to **maintain development efficiency while adopting new technology**, we can summarize that the policy Arc System Works chose for GGXrd was pretty realistic.


Collision setting tool. As with previous 2D fighting games, the collision is a 2D rectangle.



← The shader adjustment screen in Softimage, where previews allow the user to work while checking the visuals in real time.

The shader adjustment screen in UE3, where you can see that the material tree has been created to produce the same calculations as the shader created in Softimage. →



It’s Unreal Engine 3

As of the time of this report, the development team for GGXrd consists of **25 full-time members** from within Arc System Works alone.

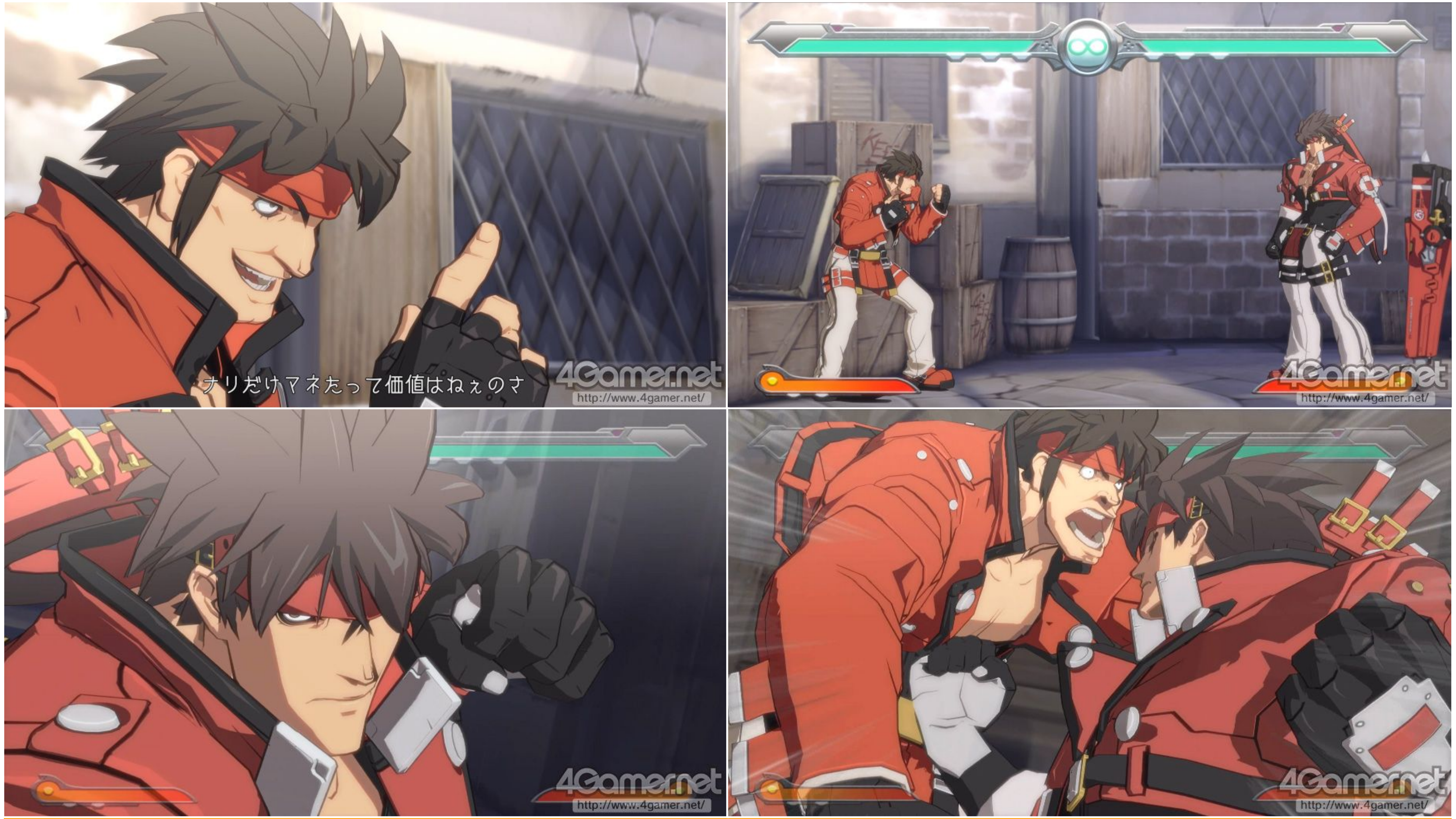
Excluding management, there are 4 programmers, designers, and 12 artists. In addition, there are outsourced staff from Japan and overseas, so the total number of people involved in the project is around 100.

The idea for the game itself was conceived around 2008, and it was launched as a concrete development project in the spring of 2011. At that time, it was almost decided that 3D graphics would be used, and a pilot movie was made for internal presentation.

Representatives of the GGXrd development team

The development of the prototype version began in the latter half of 2011, and the decision to use UE3 was made around the end of the year. Full-scale development work began in the second half of 2012, and the actual production period was about a year and a half.

It’s Unreal Engine 3



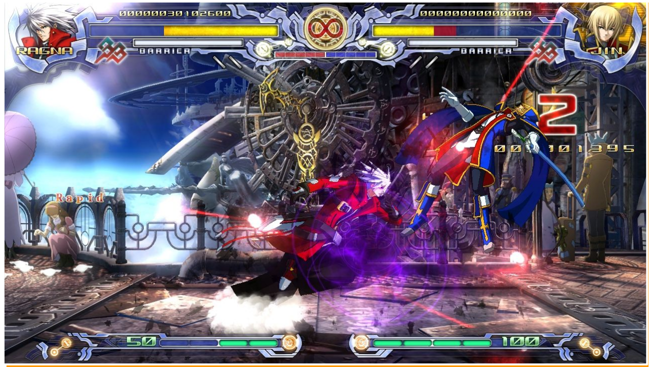
Pilot cutscene, for internal presentation.

A 2D Fighting Game with 3D Graphics

A 2D Fighting Game with 3D Graphics

Arc System Works has a proven track record of producing several 2D fighting games, and are seen as craftsmen in their area.

With the exception of the aforementioned Guilty Gear 2: Overture, the Guilty Gear series has been recognized by many gamers for its 2D pixel art. Because of that, the decision to switch to 3D graphics was a big one for Arc System Works.



BLAZBLUE © ARC SYSTEM WORKS

D. Ishiwatari—— *We started work on the new Guilty Gear in the spring of 2011. Before that, we released BLAZBLUE in 2008, which we positioned as the pinnacle of pixel 2D fighting games.*

So what could the new GG do to impact the fans? The conclusion we came to was ***real-time 3D graphics that look like pixels or cel animation****.*

Ishiwatari had not decided on 3D from the beginning, but had conducted his own research to see if vector graphics could be applied to 2D fighting games.

They had been experimenting with 3D modeling of characters from the Guilty Gear series since around 2007, but felt that there were limits to what he could do with so-called photo-realistic visuals.

Around the same time, Junya Motomura began researching toon shaders (also known as cel-shaders), which are used to create cel animation-like visuals in real time. →

A 2D Fighting Game with 3D Graphics

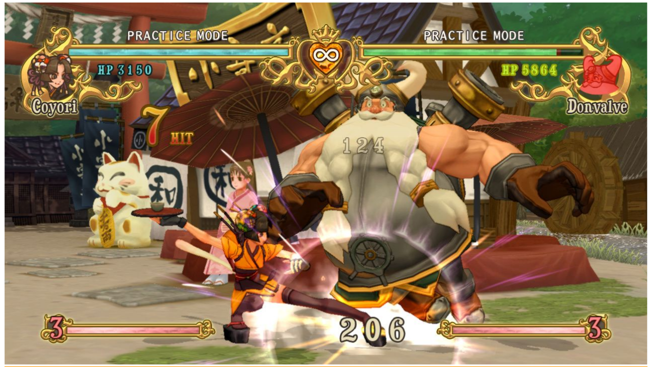
As for the programming side of things, Mr. Ieyumi said that since he had experience developing the 3D fighting game **Battle Fantasia** he wasn't too worried about adopting this direction.

Dynamic visuals can be achieved by changing the camera angle. It's easy for even a layman to see this is the main appeal of 3D, but Ishiwatari himself, as the general director, saw another great potential in this method.

D. Ishiwatari—— *If you use 3D graphics, you can make the game more interesting.*

The use of 3D graphics makes it easier to create a wide variety of facial expressions, which is difficult to achieve with pixels.

Since you can move your face independently of body movements, ***you can express a variety of emotions during battle without dialogue.***



BATTLE FANTASIA © ARC SYSTEM WORKS

Back in the days of low screen resolution, the pixelated facial expressions of characters were not as detailed as they are today. In recent years, however, as displays have increased in resolution, there is no longer much room for error in showing those details, so **when pixel art is used, you have to raise the drawing resolution to match the display resolution**.

Arc System Works had already achieved this with the BLAZBLUE series, but with GGXrd, they wanted to go beyond that and focus on 3D graphics that would allow for more flexible facial animation.

What are the Specs of Guilty Gear Xrd?

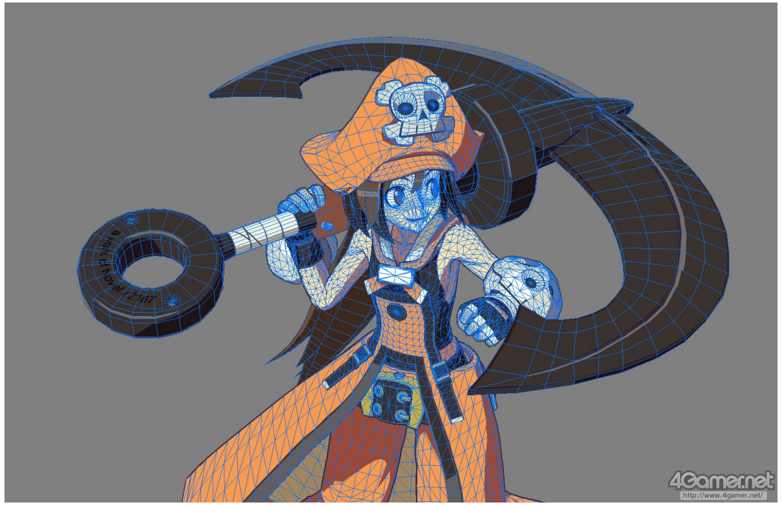
What are the Specs of Guilty Gear Xrd?

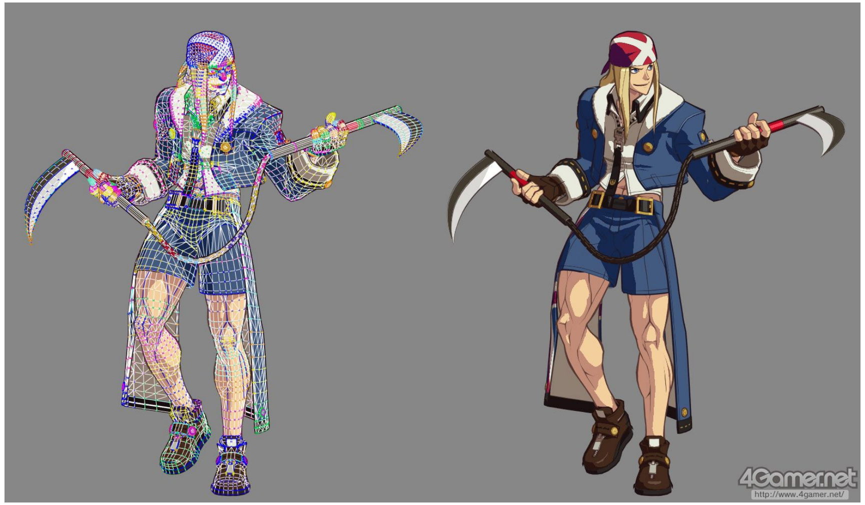
The game uses Sega's **RINGEDGE 2** (*specs not disclosed) as the system board, and since its OS is 32-bit Windows XP Embedded, the executable binary is also 32-bit.

The arcade rendering resolution is **720p at 60fps**, and instead of the standard anti-aliasing technique of MSAA (Multi-Sampled Anti-Aliasing), it uses the post-effects FXAA (Fast Approximate Anti-Aliasing).

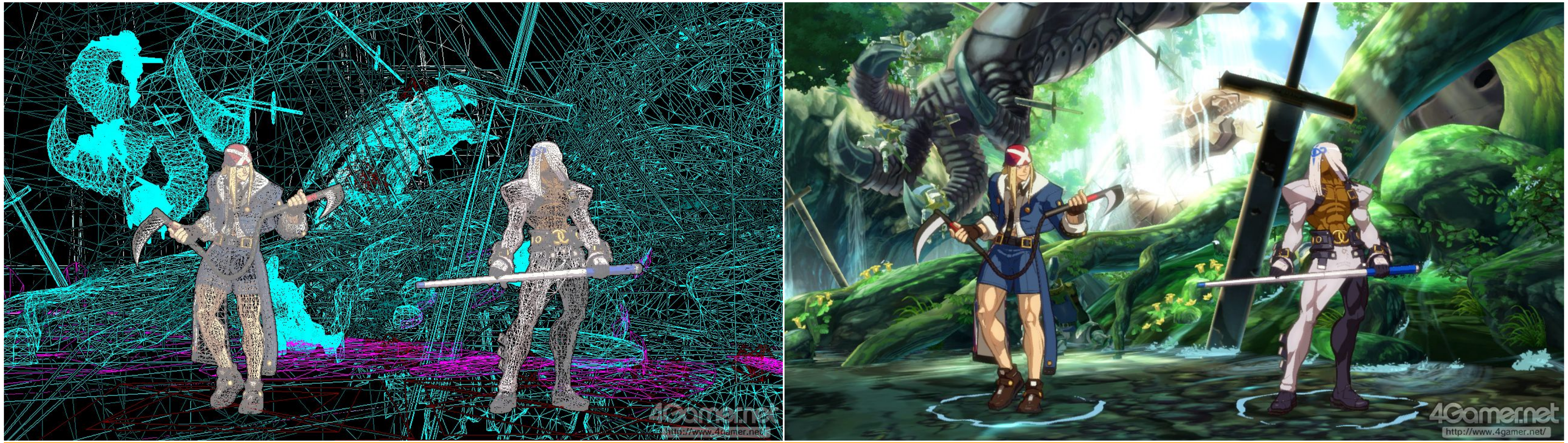
The total number of polygons per scene is about 800,000. The breakdown is 250,000 for the two characters and 550,000 for the background, but these values are for reference only, and represent the total geometry (vertex) load on the GPU, including visible and invisible polygons, when rendering a single scene.

A character model is about 40,000 polygons.





What are the Specs of Guilty Gear Xrd?



Wireframe (left) and final shot (right) of the in-game screen.

What are the Specs of Guilty Gear Xrd?

In GGXrd, the head is modeled separately for the battle scenes and the cutscenes, but the number of polygons doesn't change significantly in either scene.

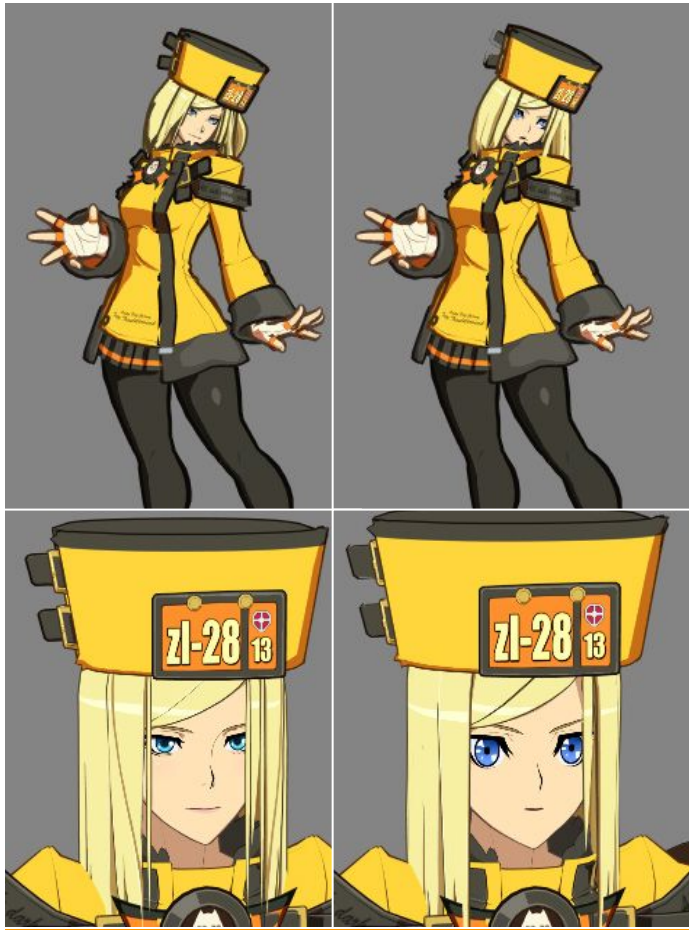
When you hear that *"3D models of the head are prepared for each character for both battle scenes and cutscenes"* people who are familiar with 3D graphics tend to imagine that there are two models, one with many polygons and another less polygons, like LoD (Level of Detail). However, Ishiwatari said that this was not the case, and that the actual reason for this was to have a good looking model for each phase of the battle.

D. Ishiwatari—— *Characters in battle will have their faces drawn small. The same goes for costumes and accessories. However, these 3D models don't fully express the identity given to the characters when they're in close-up. For this reason, we decided to create separate 3D models for battle and for use when the camera is close up.*

As an example, the series' Millia is a cool-beauty female character with long sharp eyes and a mature face. This is how she is shown in cutscenes and close-ups, like during certain special moves.

However, if the 3D model with the "mature face" is used directly in battle, the eyes and nose become too thin, making it hard to express the character's personality and expression.

So Arc System Works decided to **change the look of the models for battle and for close-ups in order to emphasize the details of the characters' expressions** and the iconic parts of their costumes and accessories. For example, in the case of Milia, the 3D model for the battle scene was intentionally changed to have larger eyes.

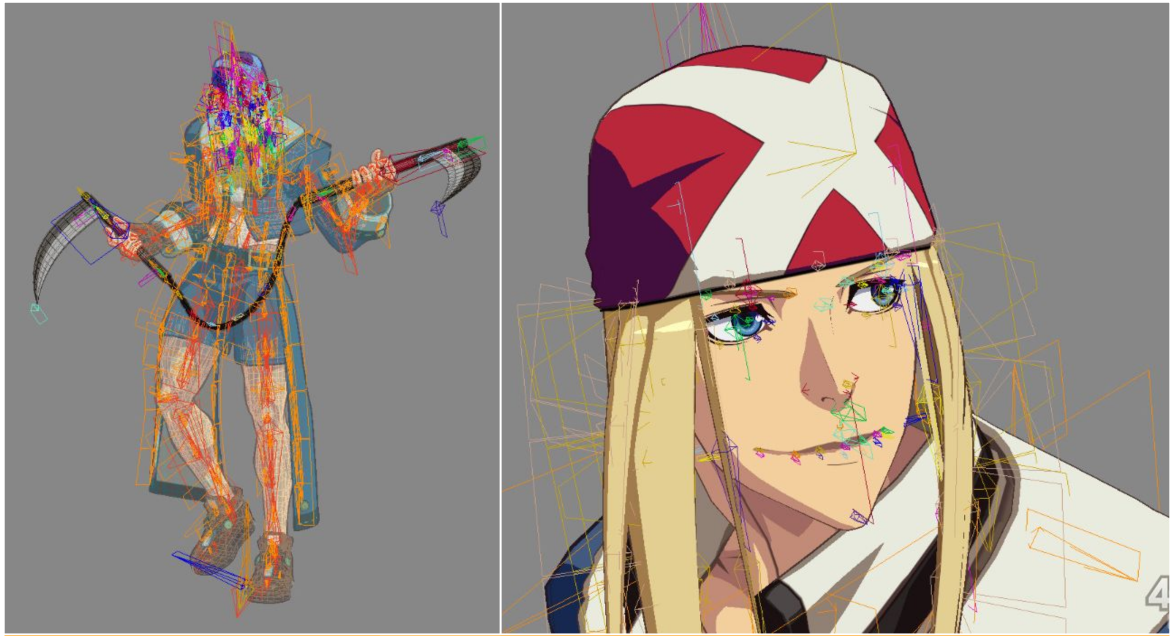


The difference between the model cutscenes (left) and for battle scenes (right). Especially noticeable in the size of the eyes.

What are the Specs of Guilty Gear Xrd?

The main character, Sol, has about 460 bones. The character with the most bones has about 600. However, the bone structure, including the limbs, is basically the same.

The number of bones in the head is about 170, of which about 70 are allocated for facial expressions. This is a lot more than the average 3D game character, and that is why they are so particular about facial expressions.


Bones visualized. Body (left) and head (right)

What are the Specs of Guilty Gear Xrd?

The rendering method is the common **Forward Rendering**, not the recently popularized Deferred Rendering, which is an effective method when dealing with a large number of light sources, but was probably not necessary for the anime-like "2D style".

However, Z-buffer rendering for early culling (*Z-Prepass, i.e. depth value advance) is used. This means that geometry rendering is done at least twice for all 3D models. In addition, another geometry rendering is done for the characters in order to add the anime-style contour lines. This will be explained later.

The total texture capacity per scene is less than 160MB, and the total number of shader programs per scene is about 60-70 vertex shaders and 70-80 pixel shaders.

All of the shaders were created within the UE3 toolset. Motomura's team was in charge of the character shaders, and the background and effects shaders were handled by each of that team’s members. It seems that there was almost no need for the programming team to develop shaders individually, and this may be the effect of the adoption of UE3.

Light and Shadow

A combination of standard toon shaders and finely tuned custom techniques.

Light and Shadow

Let us have a look at the graphics technology in GGXrd, one by one, starting with the lighting system.

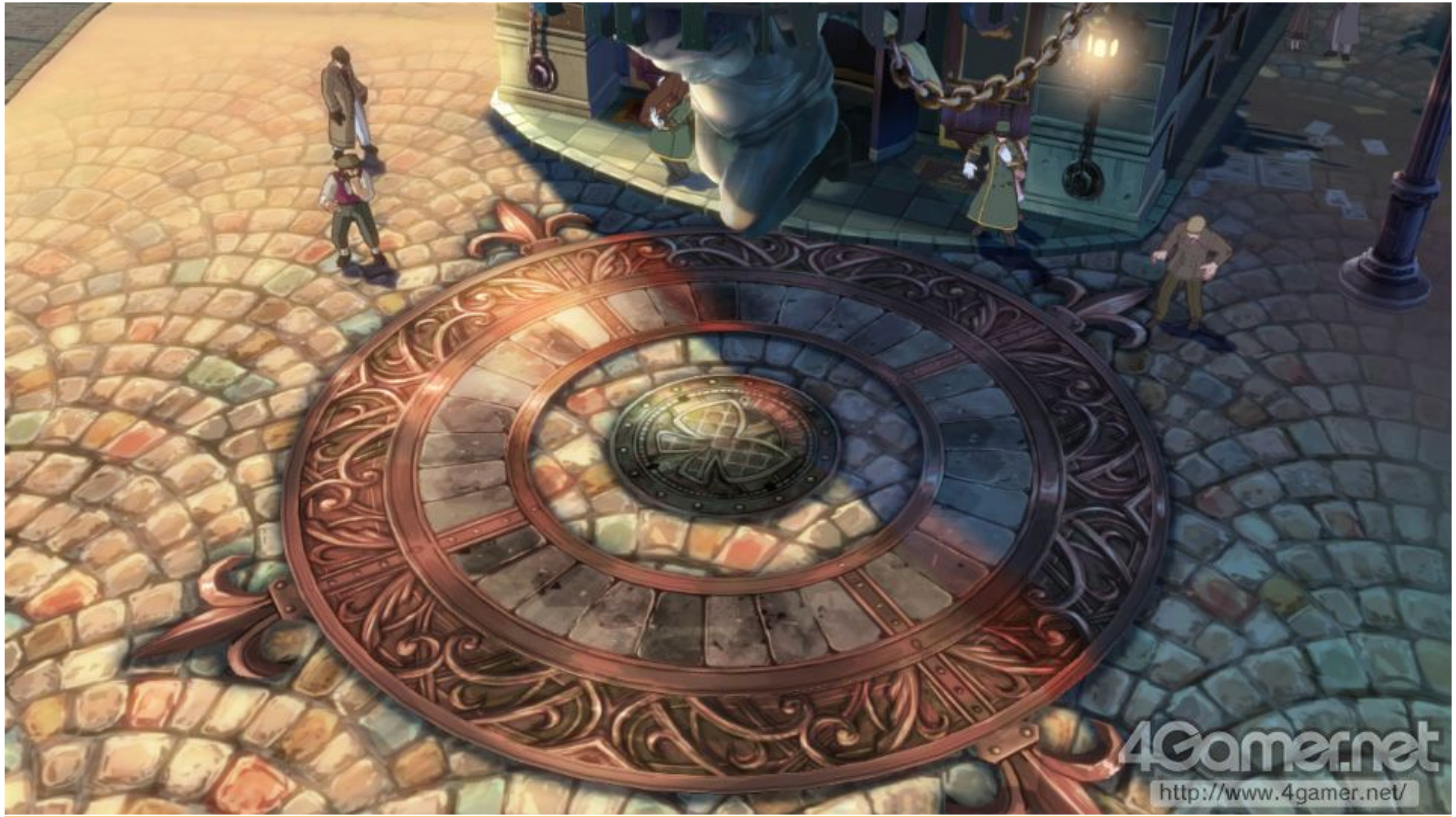
The main point of the system here is, put simply, **the complete lack of physical accuracy.**

This is related to the fact that the “ultimate goal” of GGXrd’s visuals is to emulate celluloid animation, and the development team aimed for something that was *“more artistically correct, than physically correct”*.

Looking at the game in detail, **there is only one global light source**, a parallel light source equivalent to sunlight. This light source is used to cast dynamic shadows from characters onto the ground.

Conversely, other shadows, such as those cast by static background in objects, are baked into vertex colors and textures.

Light and Shadow



Final render

Light and Shadow

Tiling Texture

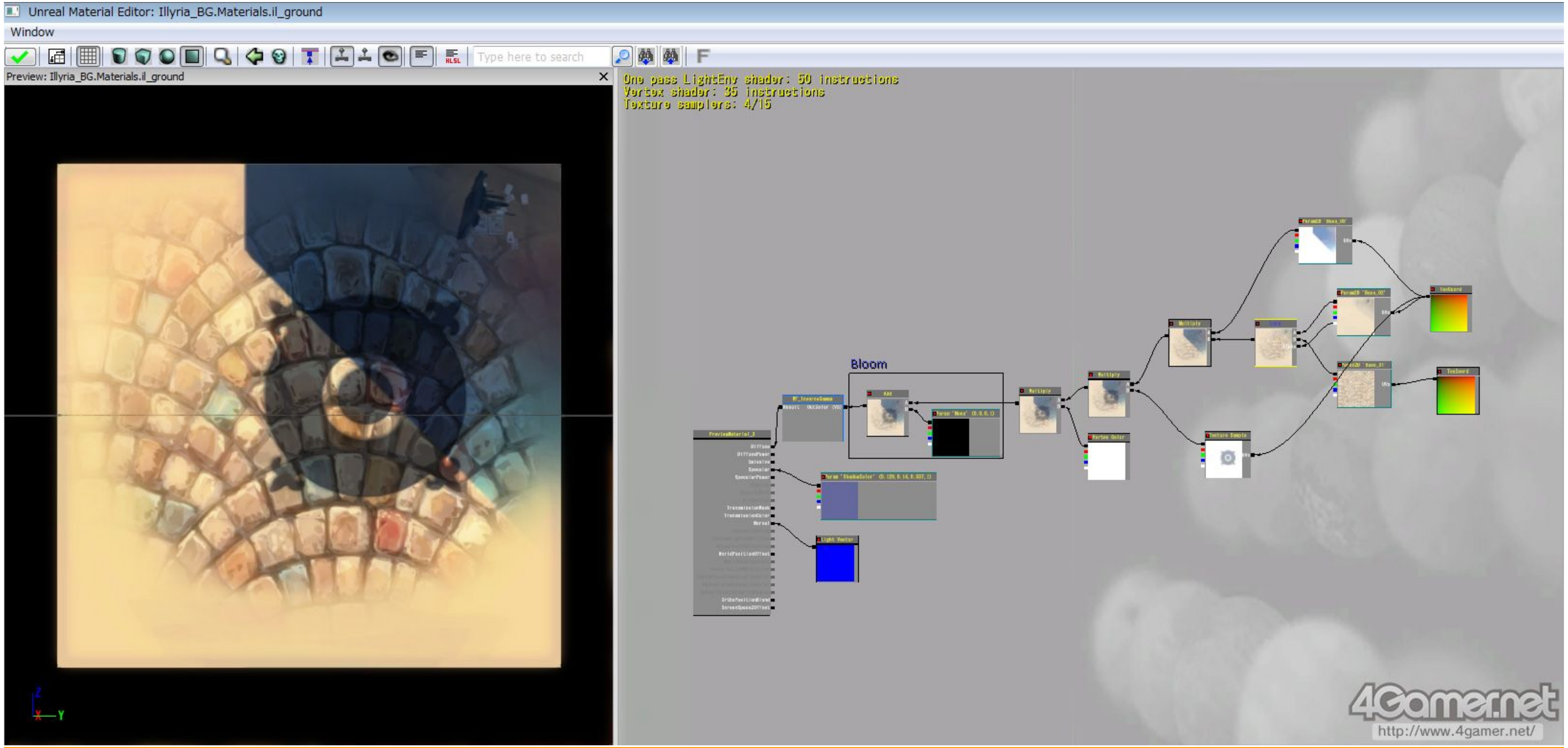
Additional Details

Shadow Texture

Manhole Material

Manhole shadow

Light and Shadow


Unreal Material Editor: Illyria_BG.Materials.il_ground
Window
Type here to search
Preview: Illyria_BG.Materials.il_ground
Bloom
4Gamer.net
http://www.4gamer.net/

One pass LightEnv shader: 50 instructions
Vertex shader: 35 instructions
Texture samplers: 4/15

Shader editing screen. The five elements shown before are combined for the final rendering result.

Light and Shadow

The shadows on the characters themselves are created by the dedicated light sources that the characters themselves walk around with. It's as if *"invisible to the player's eyes, each character's own lighting staff"* is following the character around.

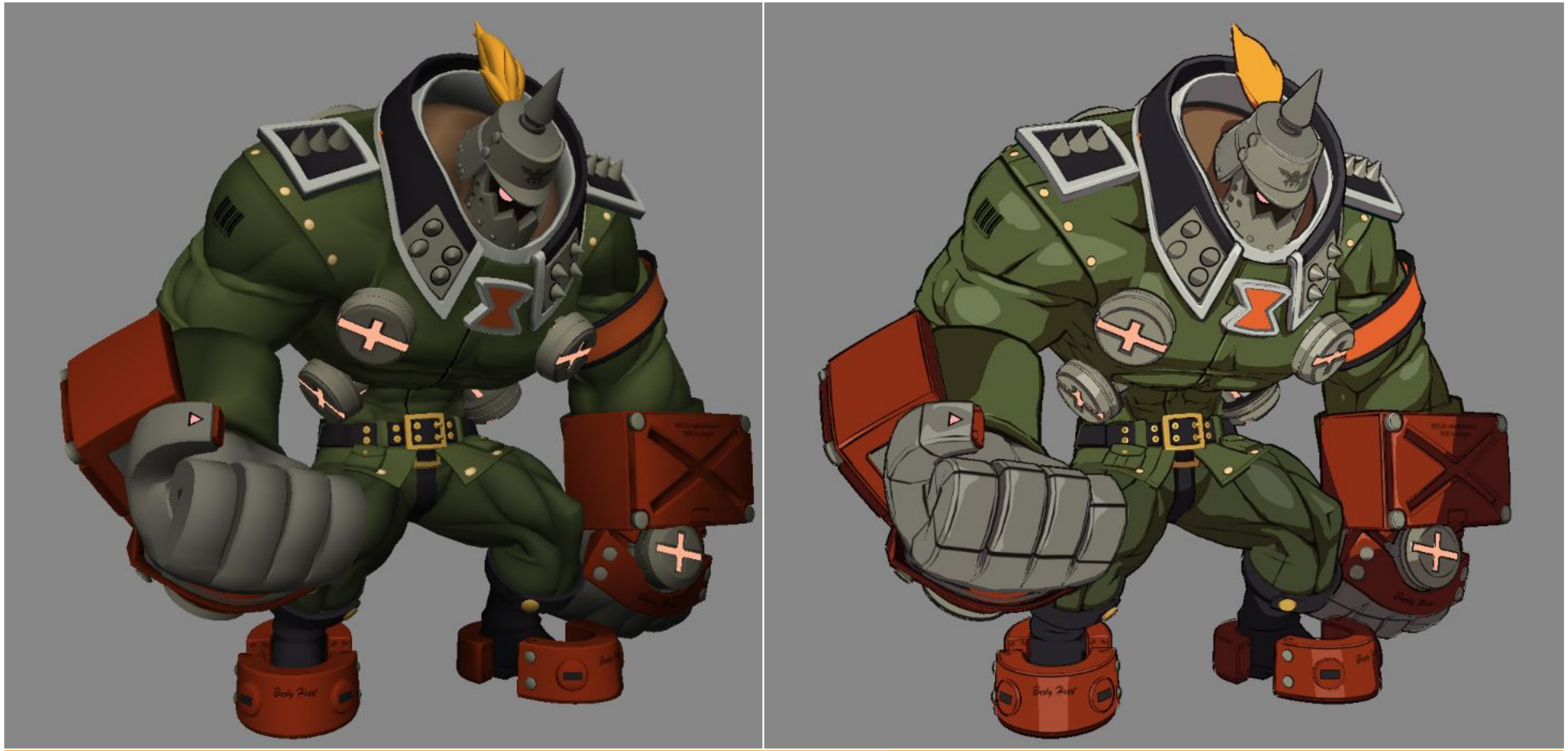
D. Ishiwatari— *The goal was to create a cel-shaded visual, which means that the emphasis was on visual appearance.*

If you light a scene with a set light source, it may look realistic in certain aspects. But when a character is placed in a certain position, for example, a shadow may fall on the face and make it look completely dark, or the shadow may disappear and make the character look flat.

This difference in appearance should be avoided in 2D fighting games, where players are always fighting on equal terms.

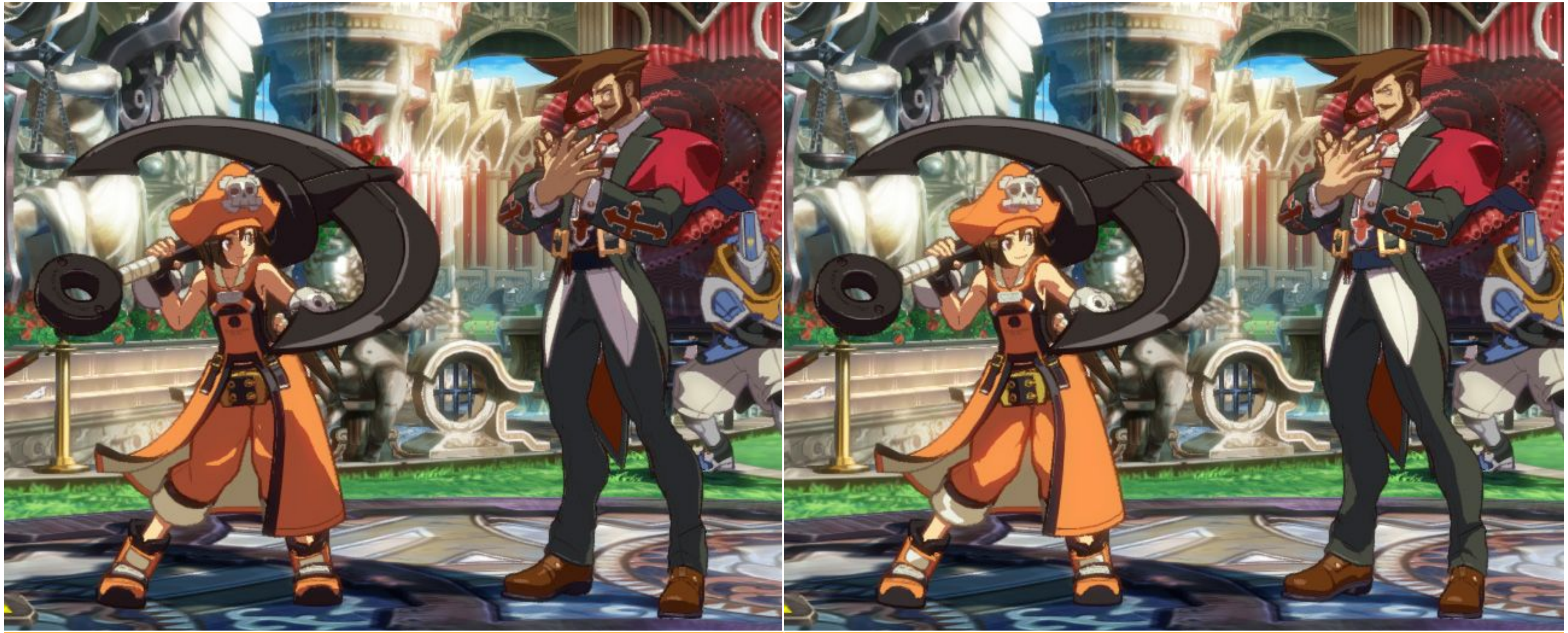
J. Motomura— *In terms of implementation, we have light sources in the lighting shader for each character. It might be more accurate to say that each character has its own vector parameters as light sources.*

Light and Shadow



Texture + Normal Shading (Left) and Cel Shading (Right)

Light and Shadow


Scene with environmental light direction (left) and with individual character lights (right)

Light and Shadow

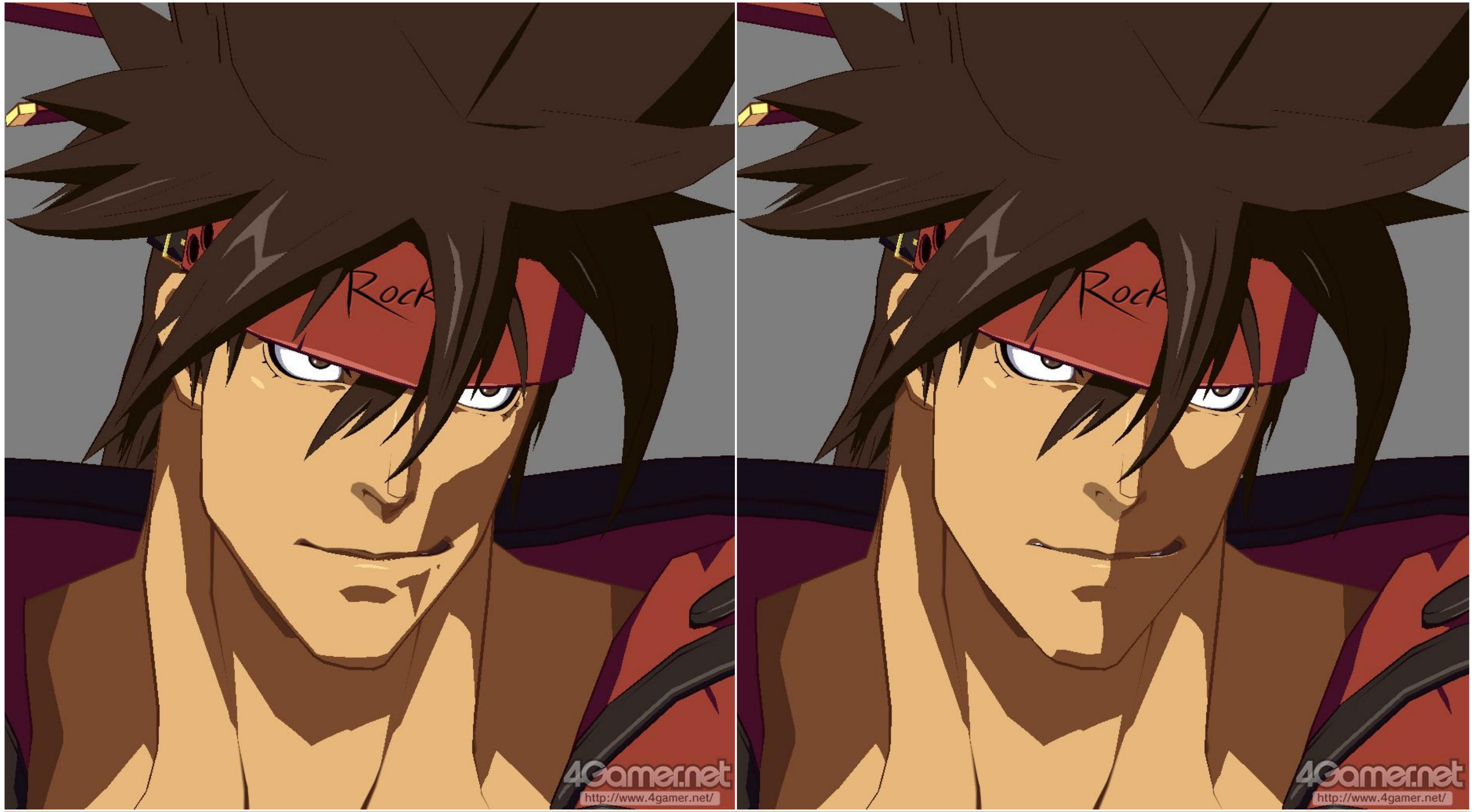
The position and angle of **this individual light source is adjusted per animation**. For cutscenes, the position and direction of the light source is adjusted frame-by-frame to make them look better.

The shading that results from the lighting is basically a very common two-toned toon shader mechanism, where there is “light” if it exceeds a certain threshold, and "dark" if it falls below. For example, if the result of lighting is a value between 0 and 255, a value above 128 is "light", and a value below 127 is "shade".

For the sake of convenience, we use "shadow" to refer to cast shadows, and "shade" to refer to areas that become darker as a result of lighting effects.

However, if the resulting values are close to the threshold, the light and shade may easily be reversed by a slight movement of the character or camera, or the areas may be broken up into small pieces and look messy, which is not a good look. →

Light and Shadow



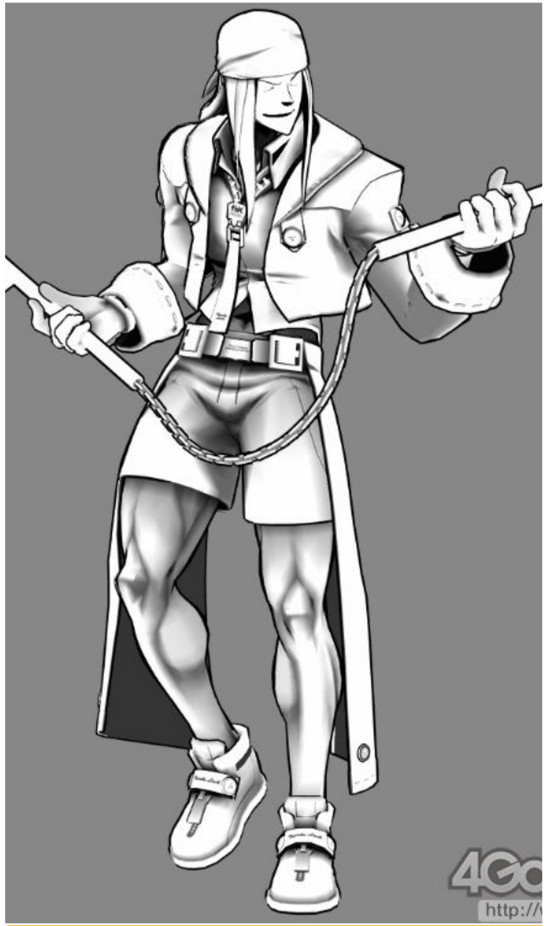
An example where the shadow shapes are broken up (left) and an example with all the corrections applied. (right)

Light and Shadow

This is why GGXrd tries to adjust the appearance of the 3D models by incorporating some innovations to make them look more like cel animation.

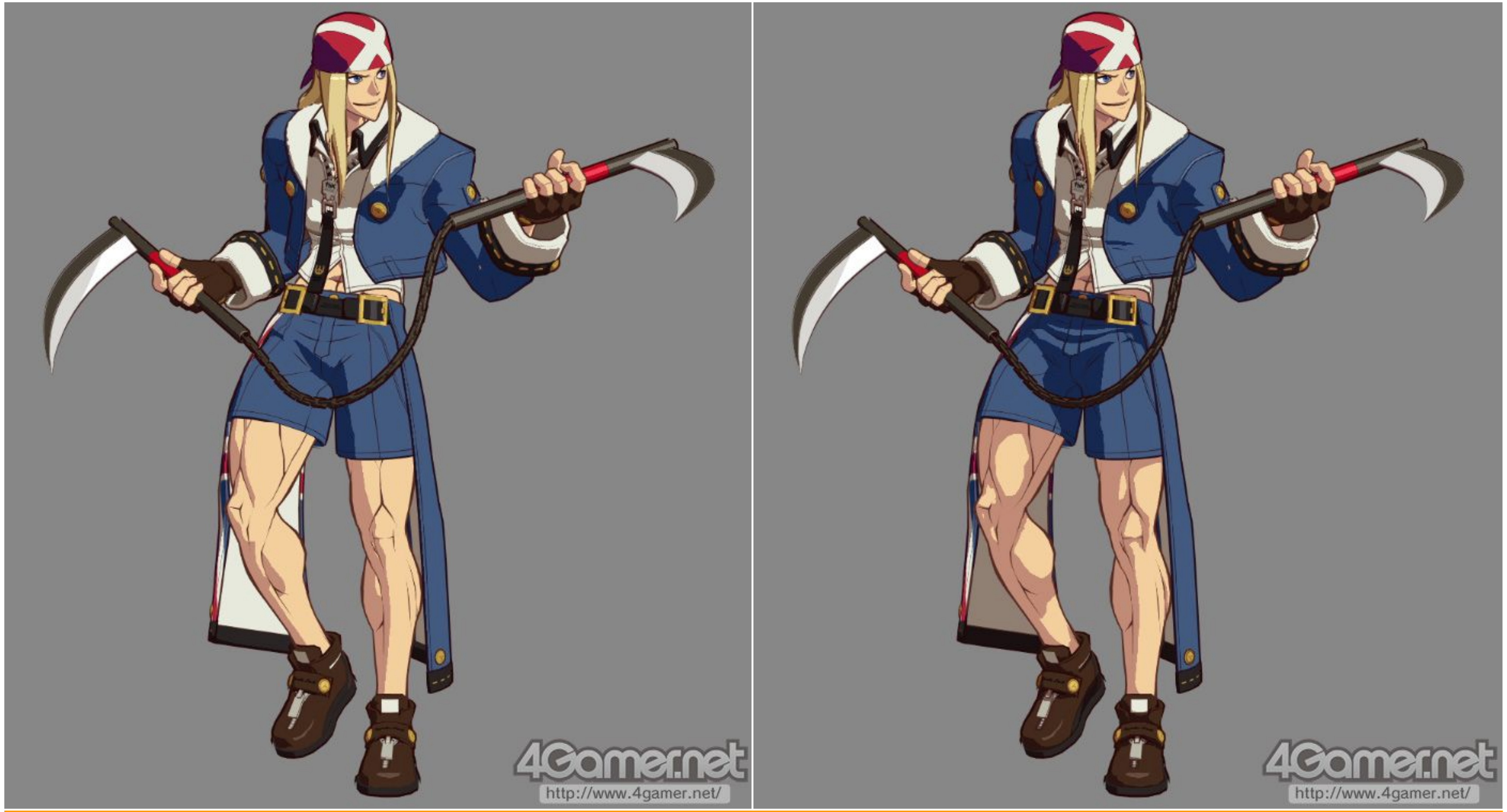
One is to add a **shade-bias parameter** to the Red channel of the vertex color, on each polygon of the 3D model.

This parameter is set to be stronger in hollow areas and areas that are occluded. In the case of the main character, Sol, this is used for the self-shadow-like shadows under the chin and around the neck, and this weight parameter contributes greatly to the shadows.



Displaying the “shade-bias” parameter. Ie.: the Red channel of vertex color.

Light and Shadow

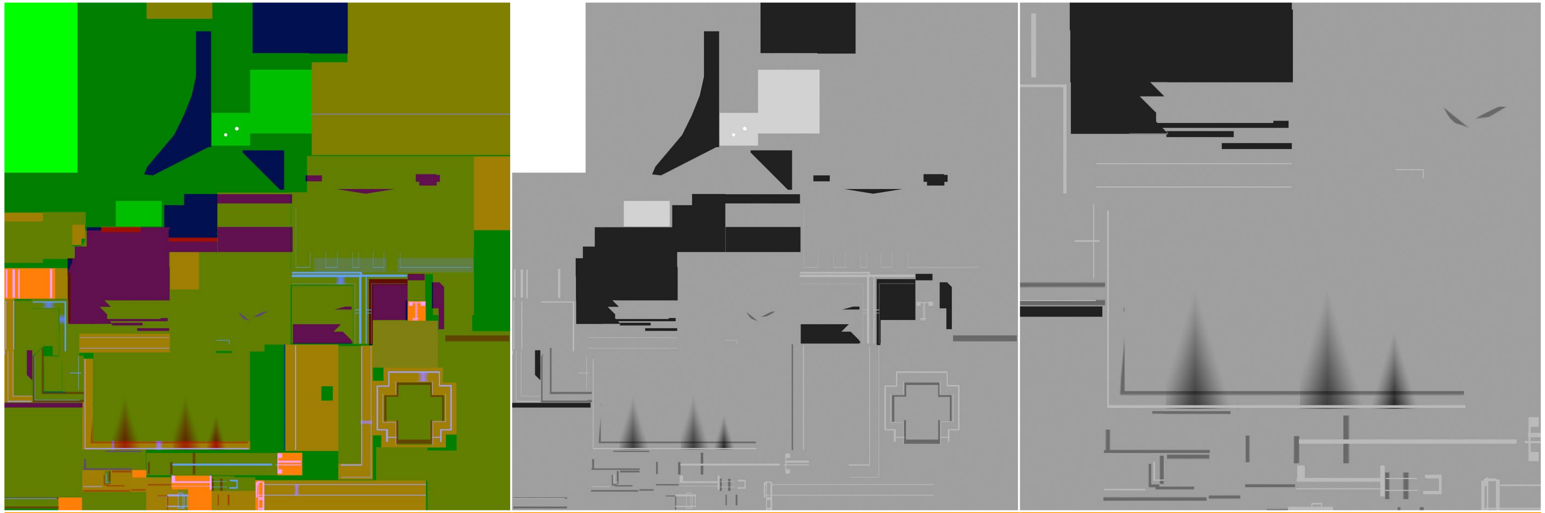


Difference between no shadow editing (left) and with shadows edited via vertex color (right)

Light and Shadow

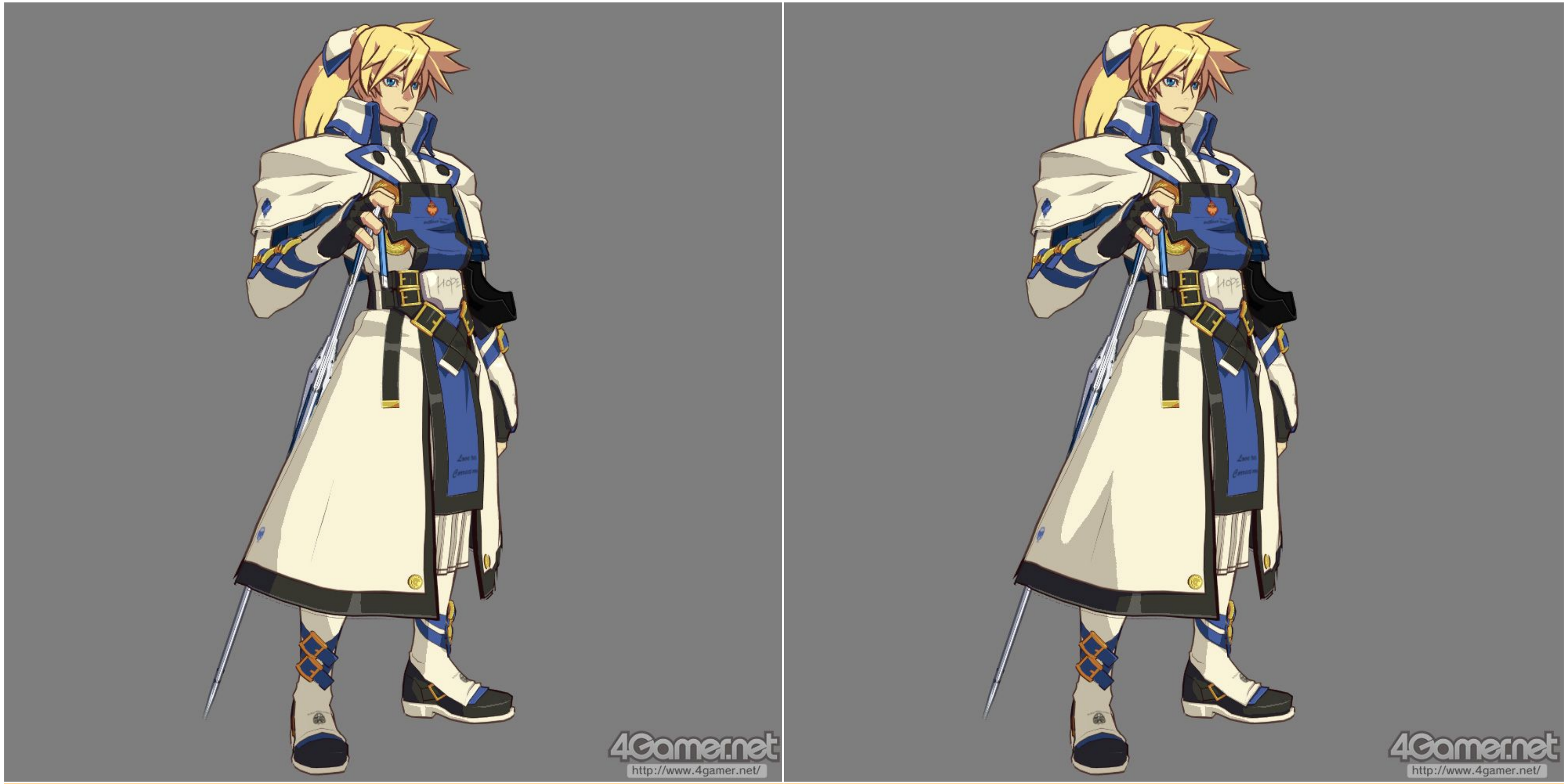
J. Motomura— The "shading-bias parameter" ends up being something like an ambient occlusion parameter, but it's set by hand based on the artist's sense.

We have a secondary "shade-bias parameter" also set in the **Green** channel of our lighting control texture, which we call the **ILM texture**. When this parameter is set to its maximum value, it creates a baked shadow. For example, we use this method to shade areas that are always heavily shaded, such as the neck area below the chin.



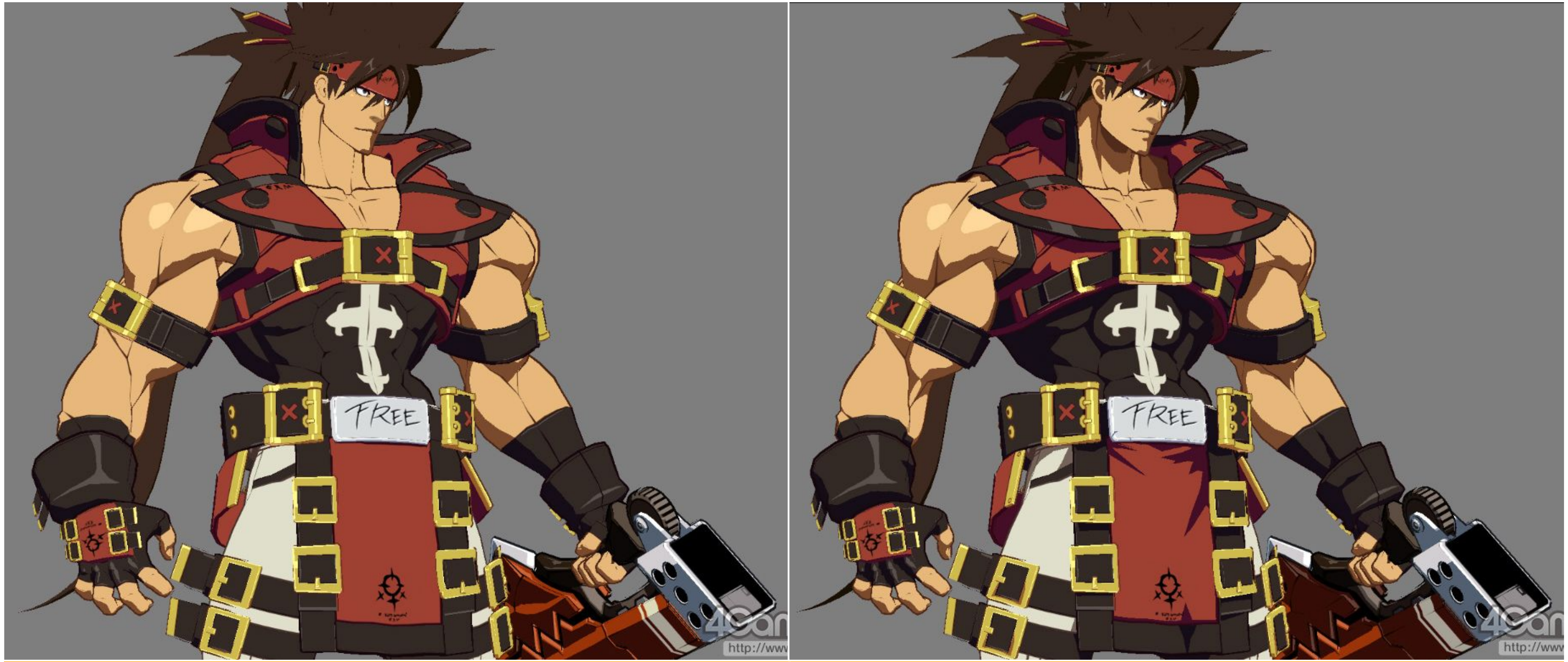
Ky's ilm texture (left) and its Green channel (center). On the right is a zoom of “skirt” area of the Green channel.

Light and Shadow



Ky before shade editing (left) and after (right).

Light and Shadow



Comparison where only light is applied without controlling the "shade-bias" parameter (left) and after shade is controlled. (right)

Light and Shadow

Another way to achieve the cell animation-like shadows is to **adjust vertex normals**. In few words words, a vertex normal, is the orientation of the vertices.

Lighting calculation in 3D graphics requires three parameters: **the direction of the light source** (light vector), **the direction of the line of sight** (view vector), and the **direction of the surface** (normal vector). In other words, **the appearance of shadows can be adjusted by changing the direction of the surface**.

In the case of GGXrd, the character models are about 40,000 polygons, as mentioned above. So if you don't do anything, you'll end up with complex shading that matches those 40,000 polygons, but what you want is rough shading that looks like cel animation. This is where the idea of ***"maintaining the 3D model shape of 40,000 polygons, and simplifying only the shadows resulting from lighting"*** comes from.

Face orientation is represented by a normal vector given to each vertex, which the team adjusted to produce simplified shading.

By default, UE3 always recalculates the vertex normals when importing rigged meshes (meshes that support bone deformation for characters). As a result, the normals edited in Softimage, a 3D graphics production tool, couldn't be reproduced in-game. →

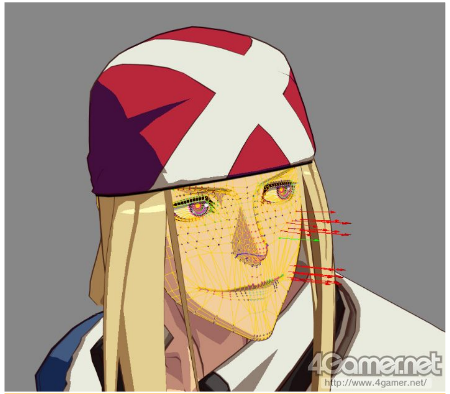
Light and Shadow

So, programmer Ieyumi modified the engine's mesh import function and so that the edited normals could be used directly in the game. *“Without this, it would have been impossible to control shading by editing normals"* recalls Motomura.

Now that UE3 supported adjusted normals, there are several options for to go about doing it. In the past, Studio Ghibli used to “flatten” the CG elements of *Howl's Moving Castle* by finding the average the values of adjacent normal vectors. What method did this team choose?

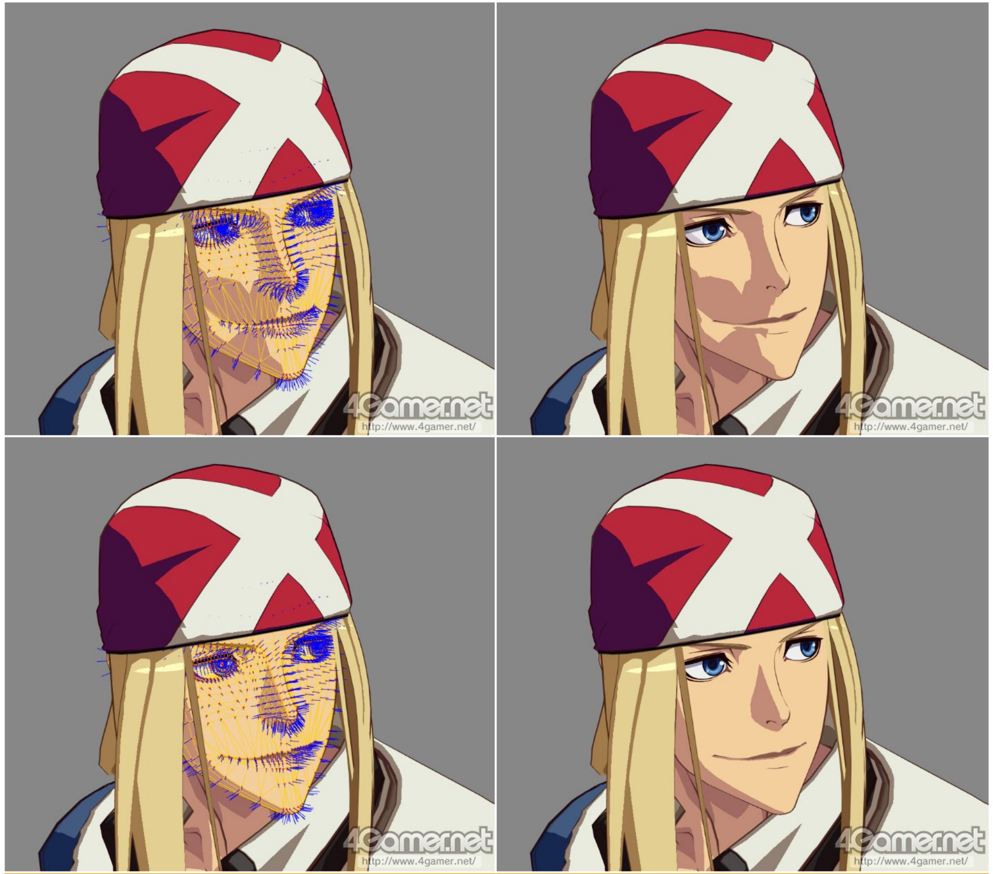
J. Motomura— For GGXrd, the appearance of the character was the most important thing, so we paid close attention to the shading on the face. For the face and head, the artists manually edited and adjusted the normals.

For example, if the normals around the cheeks are aligned closer to the normals around the temples, when the temples are dark, the shadows around the cheeks will be dark as well. The 3D model itself is modeled as a curved surface, but by aligning the normals, we can give the high-polygon model a rough shade like a low-polygon model.



The process of editing normals.

Light and Shadow



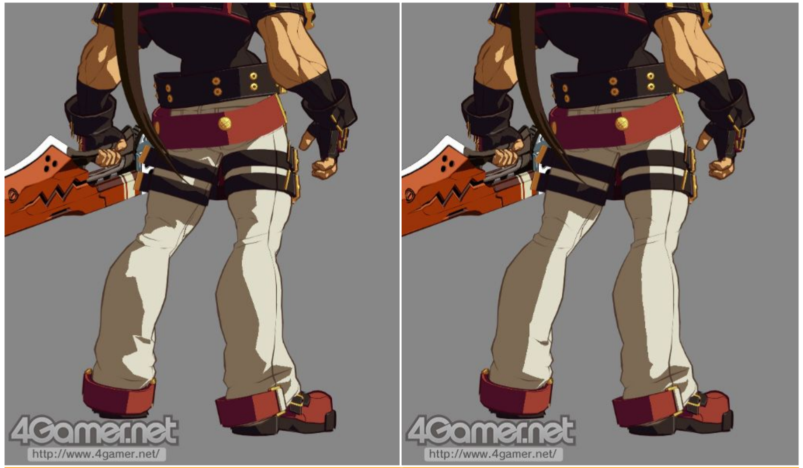
The upper row is before editing normals, the lower one after editing.

Light and Shadow

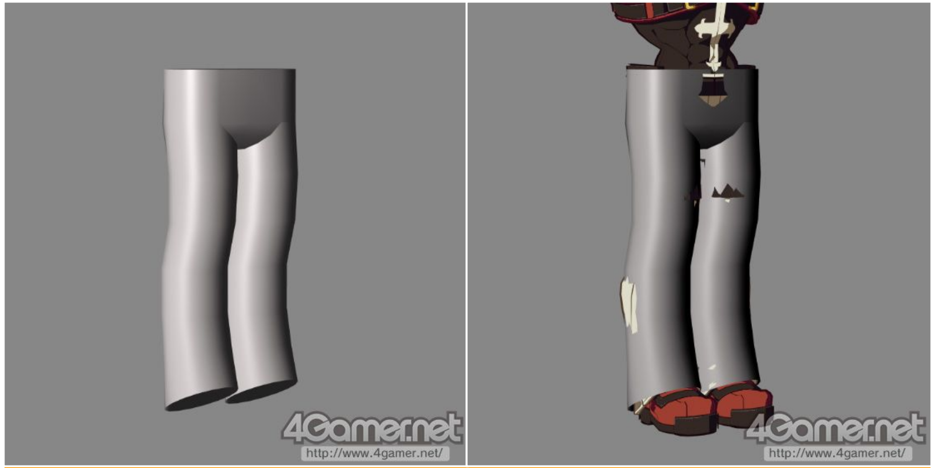
J. Motomura— In other cases, for clothing, head hair, etc., I use Gator to **transfer the normal distribution of a simpler shape model** so that a simpler shading shows on the complex shape model.

Sol's pants (or, legs) have a complex, uneven shape, but the shading is still simplified. I prepared a cylindrical model of about the same size, and transferred its normals to the pants model.

As Motomura says, this part is not done by hand, but is generated semi-automatically by using the 3D model attribute transfer function of the 3DCG software called Gator. The development team used Softimage 2013.



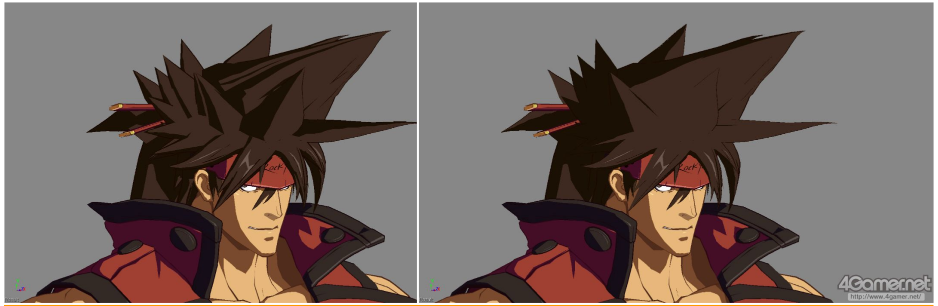
Example before (left) and after (right) using Gator for normal transfer.



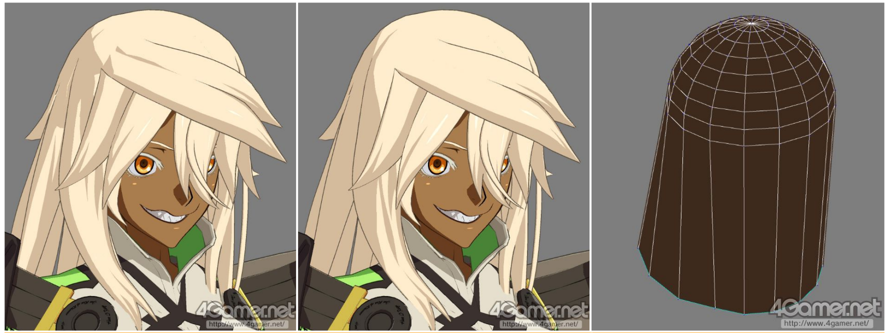
The target mesh used by Gator as a source.

Light and Shadow

The hair on the heads of the characters is quite complex, but the shading also looks correct from a cel animation perspective. If the shading were done normally, it would have been broken up into several uneven chunks, but by combining normal editing with normal transfer of simple models, and by making special adjustments during the modeling stage, they were able to achieve a good result.



Before and after editing vertex normals.



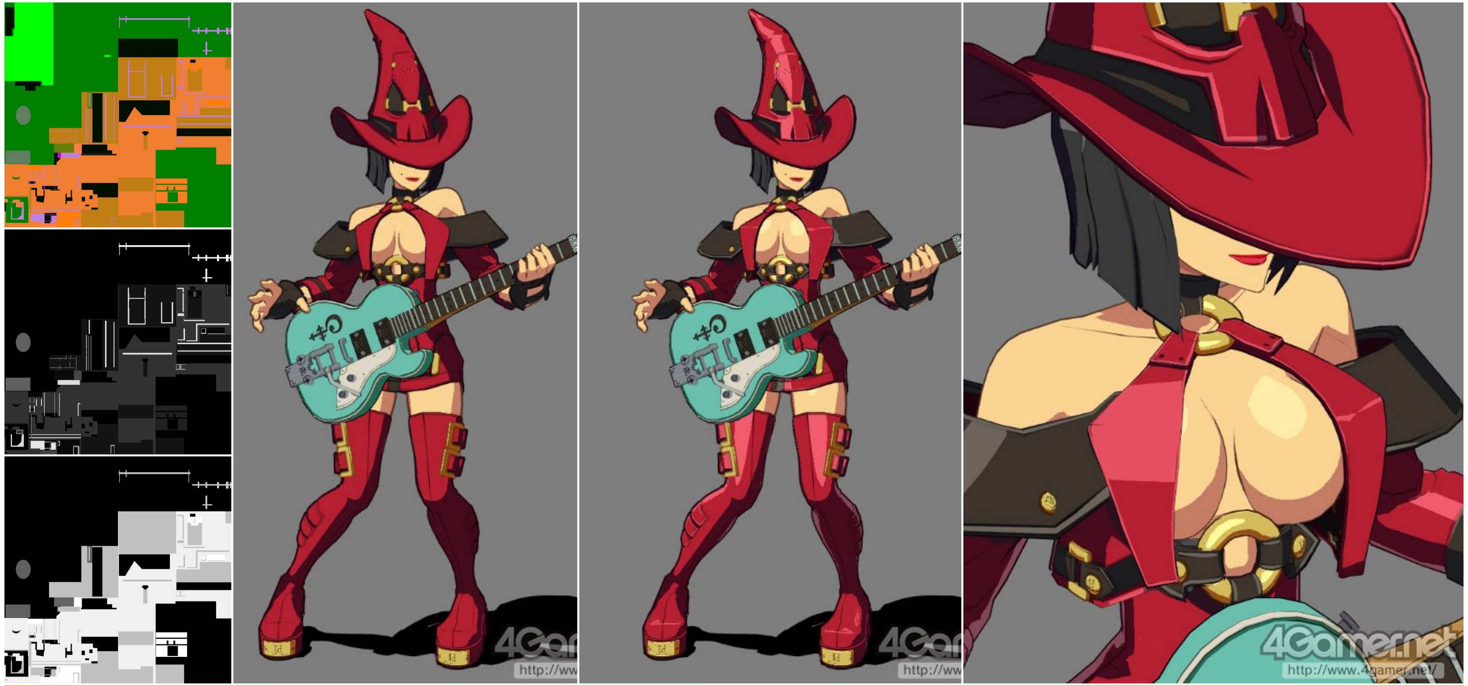
Before and After editing vertex normals, and the shape used by Gator.

Light and Shadow

Control over specular reflections and pseudo sub-surface scattering techniques.

Light and Shadow

In cel-shaded 2D graphics, specular highlights are not as commonly seen as they could be. But in GGXrd, specular highlights are added to the lighting as a result of techniques based on the language of highlights in hand-drawn illustrations.

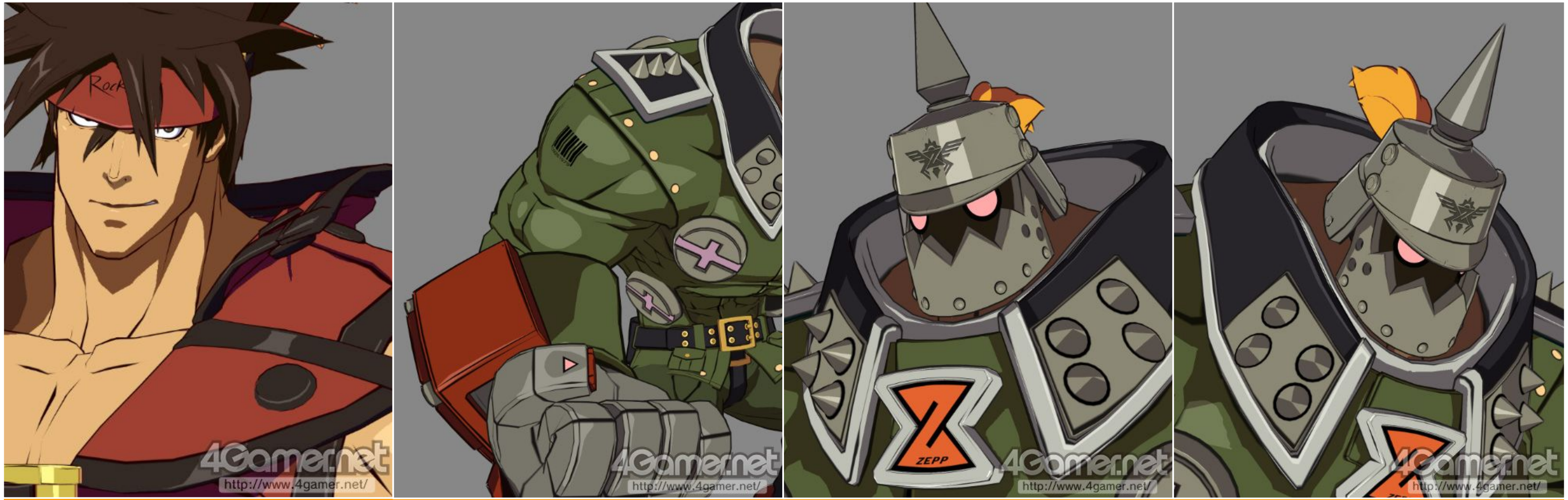


I-no's ILM texture and its Blue and Red channels. With specular disabled, enabled, and seen from a different angle.

Light and Shadow

So what is the *"language"*?

It's hard to describe in a few words, but to give some examples, it's like making sure that highlights don't stick to each other when they are closeby, but made of different materials, like skin and clothes, or making sure that highlights follow uneven or material boundaries even though they are physically incorrect.



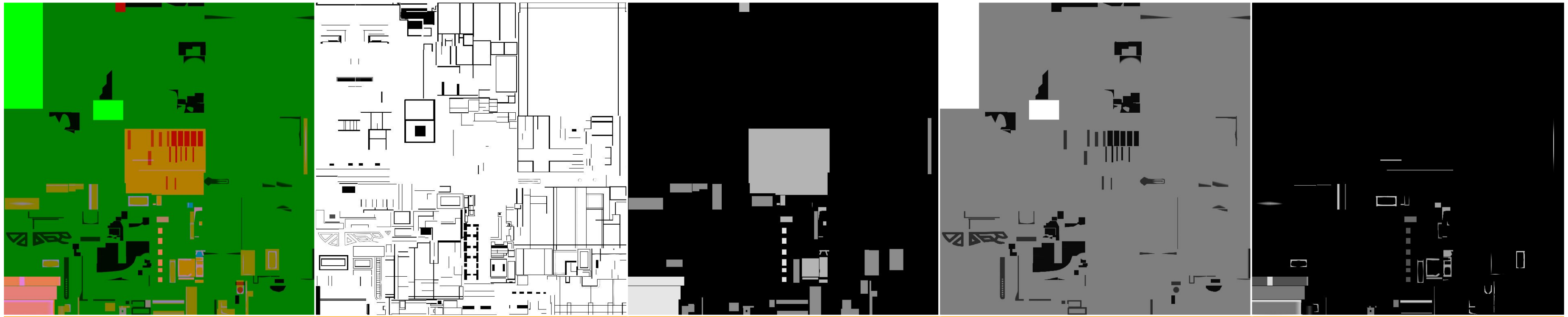
(1) The highlights on the chest, hair, lower edge of the headband, cheeks, and lips are mostly fixed, but they change in thickness depending on the angle and disappear at extreme angles.
(2) For Potemkin, the texture is set to highlight the muscles even on top of clothes.
(3) The metallic sheen changes with the angle, as it would look strange if it were fixed.
(4) The angle of the camera and the light makes the highlights always appear in the best position, which makes the image more convincing.

Light and Shadow

Specular highlighting is controlled by the "highlightability" parameter, which is stored in the **Blue** channel of the ilm texture. This parameter adjusts the intensity of the specular reflections, so the maximum value always results in baked-in highlights, and conversely, the smaller the value, the more likely the highlights will fade.

Meanwhile, the **Red** channel of the texture for lighting control is the "specular highlight intensity" parameter, and larger values are set for areas of metal and slippery materials.

However, even with all these innovations, the development team was still not convinced. What were they missing to achieve their goal of a *"perfect cel animation-like"* flavor? →


An ILM Texture and it's several channels displayed. Alpha channel, Red Channel, Green Channel and Blue Channel.

Light and Shadow

J. Motomura— *Color. It's the color.*

Light and dark are created as a result of the toon shader, but I felt that gave a somewhat boring look. A simple cel shader process that simply multiplies the shaded areas by a single shade of color would lack the credibility and richness of the material.

*But, in TV animation production, **there are artists who specialize in color design, and they set the colors individually in detail**, saying "this part of the character is this color when the sun shines on it, and this color when the character is in the shade". This is where I thought the difference might lie.*

Of course, that's not possible in this case. But as a result of his research for systematic implementation, he came to the conclusion that the color designers of TV animation instinctively decide the colors to be set by examining the **ambient light color of the scene** and the **light transmittance of the material to be expressed**. Motomura said that when he tried to implement the system based on this theory, the results were quite close to the ideal, and he decided to include it in the final specs.

The actual system is not too complicated. First you prepare a "Shadow Tint" texture that corresponds to the base texture applied to the 3D model. The team calls this texture the "SSS texture" (SSS: SubSurface Scattering) for convenience.

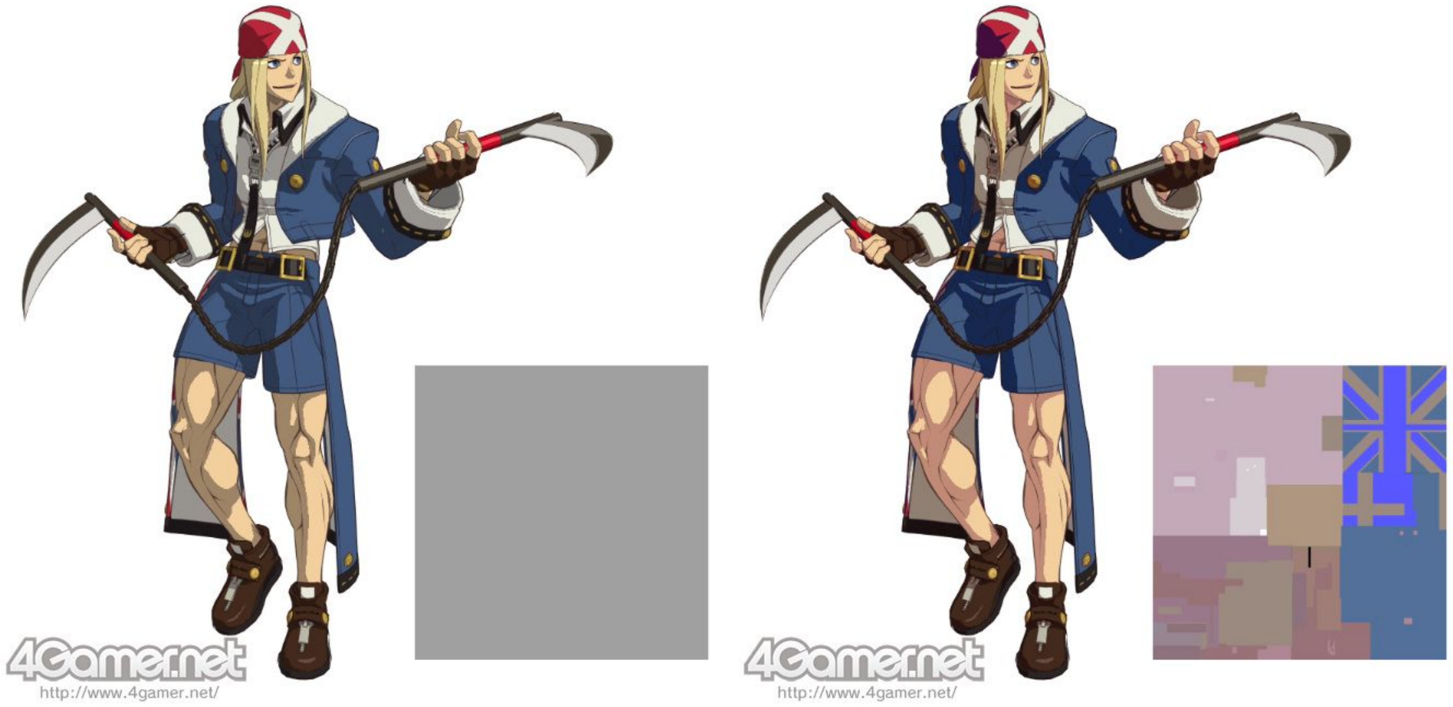
If the lighting results in a shade, then the shader multiplies the value of the Base Texture by the value of the SSS Texture to obtain the pixel color. If the lighting results in a lit area, the SSS texture value is ignored, so only the light source color is affected.

The development team was satisfied with the results of this separation process, which brought the colors closer to the cell animation style.

Junya C. Motomura (Lead Modeler and Technical Artist) is in charge of overall character model creation and bone and rig design for GGXrd. His past works include the BlazBlue series, the fighting game version of the Persona 4 series, and Guilty Gear 2: Overture.

Light and Shadow

For example, the shade on the skin tone of the character will have **a slight reddish tint**, and the shade on the clothes will retain the saturation of the color of the clothes. In other words, SSS textures are composed of such "red tones" and "colors with remaining clothing saturation.

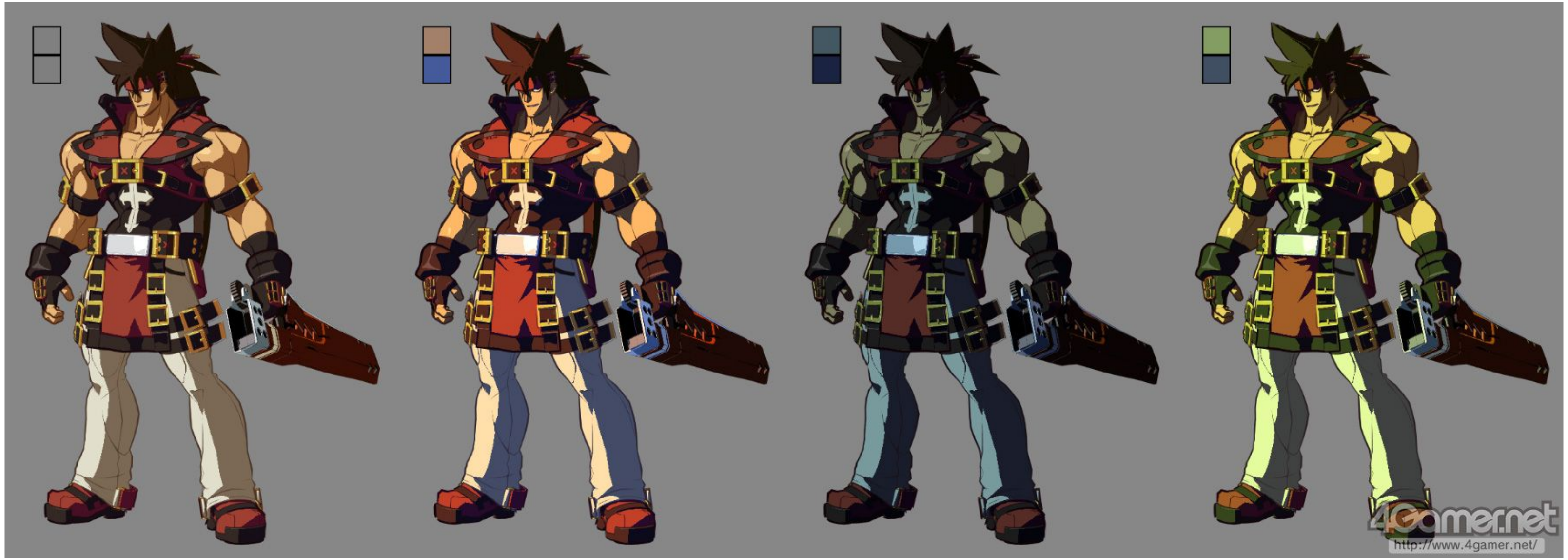


Comparison between without (left) and with (right) the SSS texture.

Light and Shadow

J. Motomura— *SSS textures do not simulate subsurface scattering, so the name may not be strictly correct (laughs).*

If I had to give a supplementary explanation, I would say that SSS textures simply show "how much light is transmitted through a material. Shadows on a thin sheet of paper are lighter in color, right? That's the kind of image you get with this texture.



Color adjustments via ambient light and light source.

TN.: This image in particular is curious, as I don't think Guilty Gear Xrd -SIGN- made use of these color adjustments.
They would later be visibly put into use in the following title, Guilty Gear Xrd -REVELATOR-.

The Secrets Behind the Linework

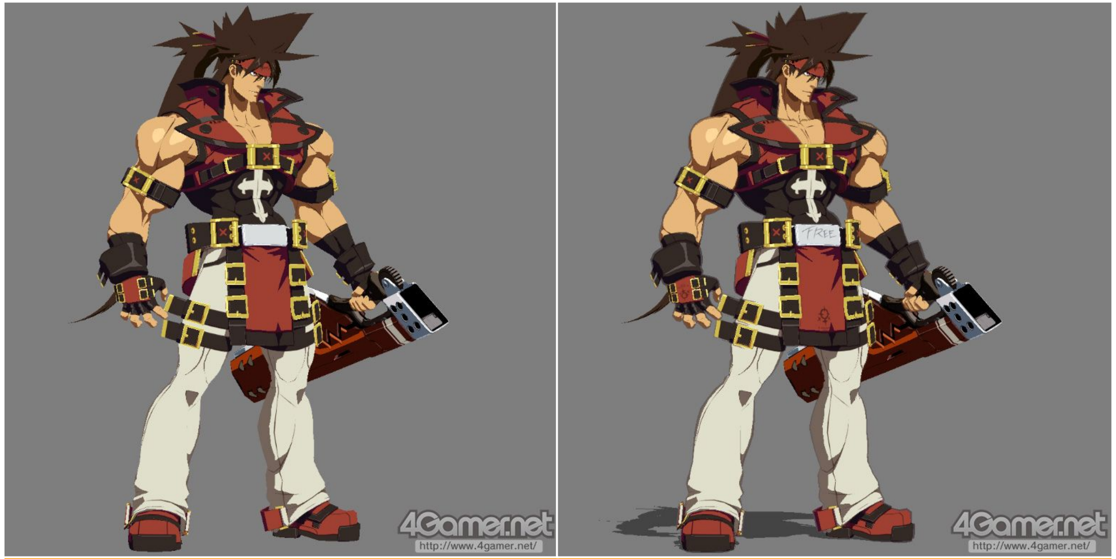
The inverted hull method.

The Secrets Behind the Linework

One of the most important elements of GGXrd's anime-style visuals are the outlines. Two approaches are combined to create these lines. The **inverted hull method** is used for the most basic line drawing of the 3D models.

Normally, when a 3D model is drawn on the GPU, polygons on the back side of the model are discarded as "invisible" and are not drawn. This mechanism is called "**backface culling**," which is based on the idea that the polygons on the back of a front-facing character model are not visible from the viewpoint anyway, so they are not drawn.

Here, backface culling is combined with flipping the faces of the polygons.



Without and with outlines.

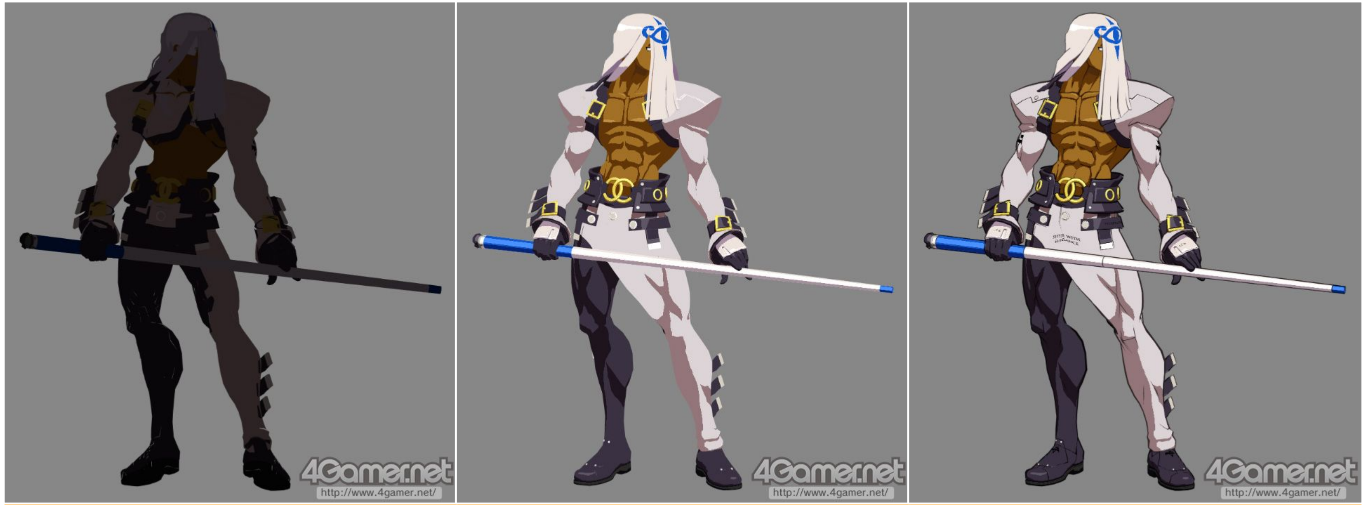
The Secrets Behind the Linework

To explain the process, the first step is to generate the hull, a process which consists of slightly inflating and inverting the 3D model. This results in a pitch-black silhouette of the 3D model being drawn, which will be saved for now.

The second step is to render the 3D model in its original size using a standard process.

Lastly, the pitch-black silhouette and the normal rendering result are combined. Most of the silhouette is overwritten by the regular render, but since it’s a slightly inflated 3D model, only the outline remains as a result.

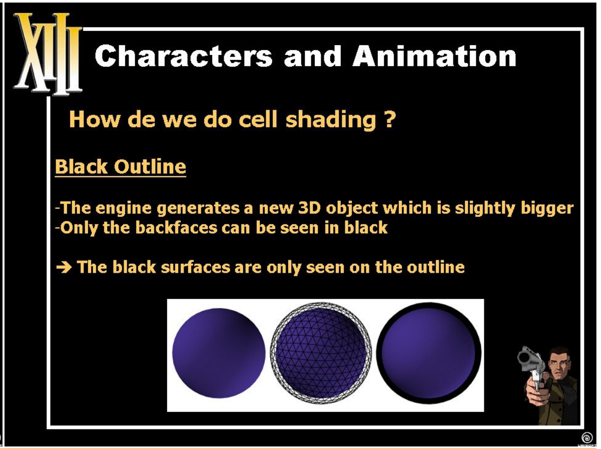
GGXrd's rendering pipeline uses the Z-buffer first, so the contour lines are nearly perfect at the first stage. So the final compositing phase may seem unnecessary, but the concept is as follows.



The rendered inverted hull, normally rendered character model and the final image. There's another secret to this, but we'll get to that later.

The Secrets Behind the Linework

In fact, this is a classic technique that has been used since before the rise of programmable shader technology, but Arc System Works uses vertex shaders to implement a unique extension of this classic technique.


XIII

Characters and Animation

How de we do cell shading ?

Black Outline

-The engine generates a new 3D object which is slightly bigger
-Only the backfaces can be seen in black

➔ The black surfaces are only seen on the outline

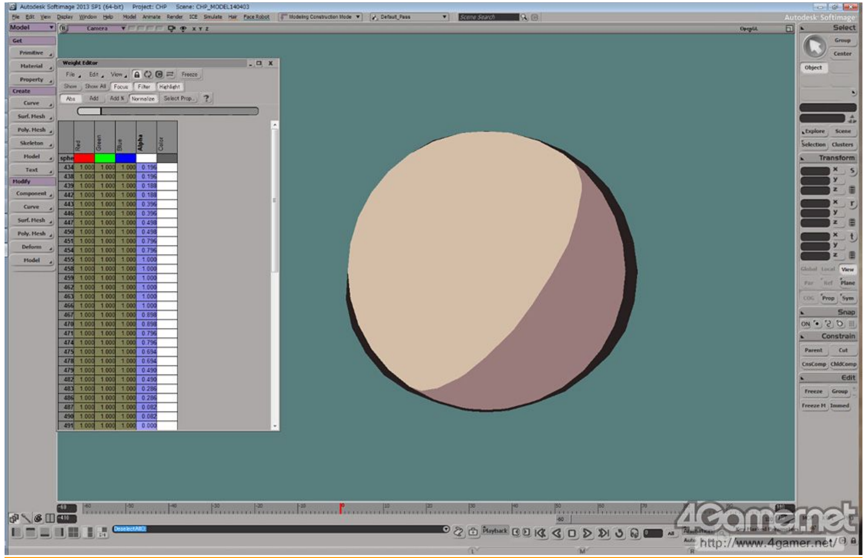
Contour generation using the inverted hull method was also used in 3D game graphics before programmable shaders. This slide is from "XIII" (Ubisoft, 2004), which utilizes the technique. The original was for PlayStation 2, and of course did not utilize programmable shaders.

Their original additions include **controls to prevent lines from becoming too thin or too thick** regardless of camera zoom or character perspective, as well as **controls to increase or decrease the thickness of lines in curved and straight areas to reproduce the feel of hand-drawn drawings**. The lines in GGXrd look as if they were drawn with a real pen, and are the result of this kind of vertex shader control.

The Secrets Behind the Linework

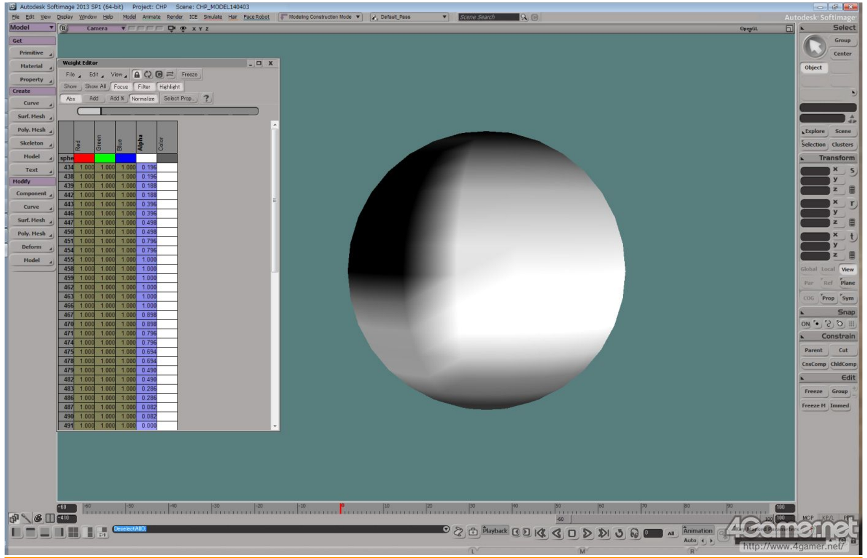
J. Motomura— *We adopted the inverted hull method because we felt it had the advantage of allowing the artist to control the width of the lines.*

In the 3D model, we have a "line thickness control value" in the vertex color, which allows the artist to freely control the width of line drawing. This makes it possible to create styles such as those seen in hand-drawn animation, where the cheeks are thicker and the chin becomes thinner.



sphe	Red	Green	Blue	Alpha	Color
434	1.000	1.000	1.000	0.196	
438	1.000	1.000	1.000	0.196	
439	1.000	1.000	1.000	0.188	
442	1.000	1.000	1.000	0.188	
443	1.000	1.000	1.000	0.396	
446	1.000	1.000	1.000	0.396	
447	1.000	1.000	1.000	0.498	
450	1.000	1.000	1.000	0.498	
451	1.000	1.000	1.000	0.796	
454	1.000	1.000	1.000	0.796	
455	1.000	1.000	1.000	1.000	
458	1.000	1.000	1.000	1.000	
459	1.000	1.000	1.000	1.000	
462	1.000	1.000	1.000	1.000	
463	1.000	1.000	1.000	1.000	
466	1.000	1.000	1.000	1.000	
467	1.000	1.000	1.000	0.898	
470	1.000	1.000	1.000	0.898	
471	1.000	1.000	1.000	0.796	
474	1.000	1.000	1.000	0.796	
475	1.000	1.000	1.000	0.694	
478	1.000	1.000	1.000	0.694	
479	1.000	1.000	1.000	0.490	
482	1.000	1.000	1.000	0.490	
483	1.000	1.000	1.000	0.286	
486	1.000	1.000	1.000	0.286	
487	1.000	1.000	1.000	0.082	
490	1.000	1.000	1.000	0.082	
491	1.000	1.000	1.000	0.000	

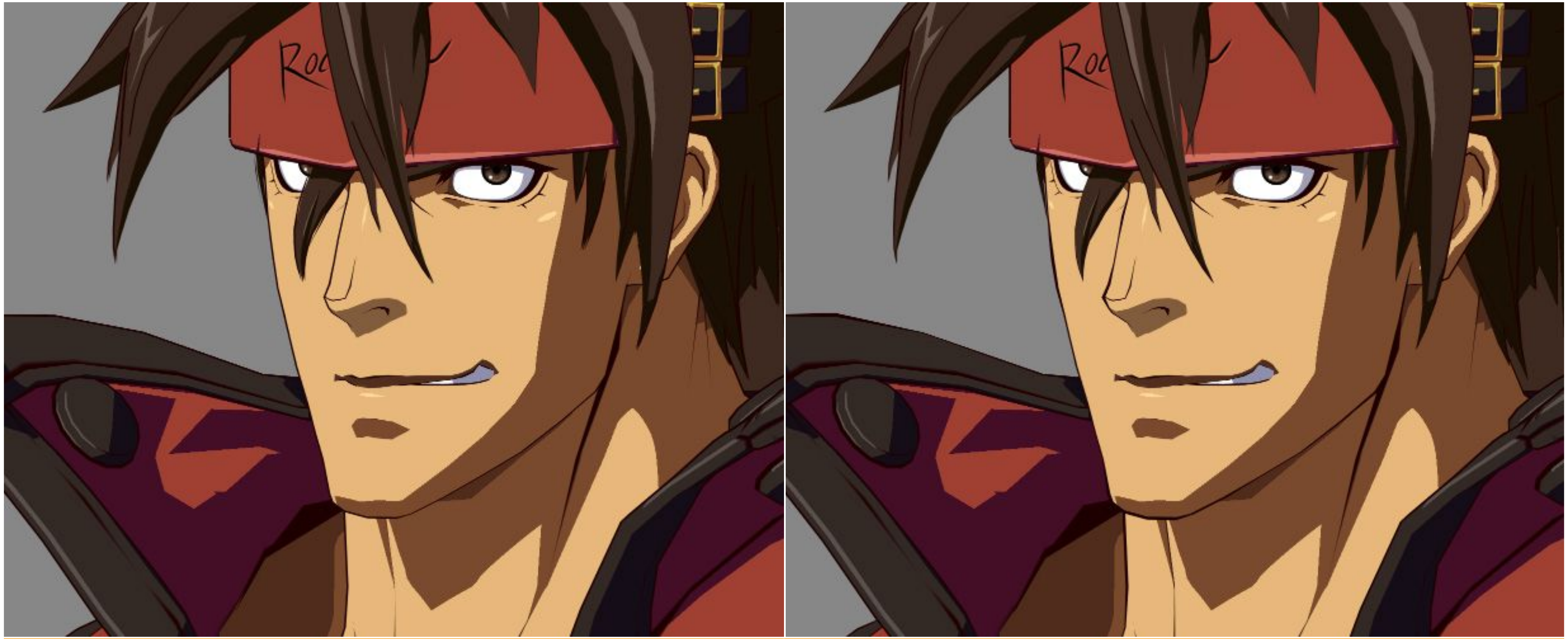
Result of adjusting the thickness of the outline.



sphe	Red	Green	Blue	Alpha	Color
434	1.000	1.000	1.000	0.196	
438	1.000	1.000	1.000	0.196	
439	1.000	1.000	1.000	0.188	
442	1.000	1.000	1.000	0.188	
443	1.000	1.000	1.000	0.396	
446	1.000	1.000	1.000	0.396	
447	1.000	1.000	1.000	0.498	
450	1.000	1.000	1.000	0.498	
451	1.000	1.000	1.000	0.796	
454	1.000	1.000	1.000	0.796	
455	1.000	1.000	1.000	1.000	
458	1.000	1.000	1.000	1.000	
459	1.000	1.000	1.000	1.000	
462	1.000	1.000	1.000	1.000	
463	1.000	1.000	1.000	1.000	
466	1.000	1.000	1.000	1.000	
467	1.000	1.000	1.000	0.898	
470	1.000	1.000	1.000	0.898	
471	1.000	1.000	1.000	0.796	
474	1.000	1.000	1.000	0.796	
475	1.000	1.000	1.000	0.694	
478	1.000	1.000	1.000	0.694	
479	1.000	1.000	1.000	0.490	
482	1.000	1.000	1.000	0.490	
483	1.000	1.000	1.000	0.286	
486	1.000	1.000	1.000	0.286	
487	1.000	1.000	1.000	0.082	
490	1.000	1.000	1.000	0.082	
491	1.000	1.000	1.000	0.000	

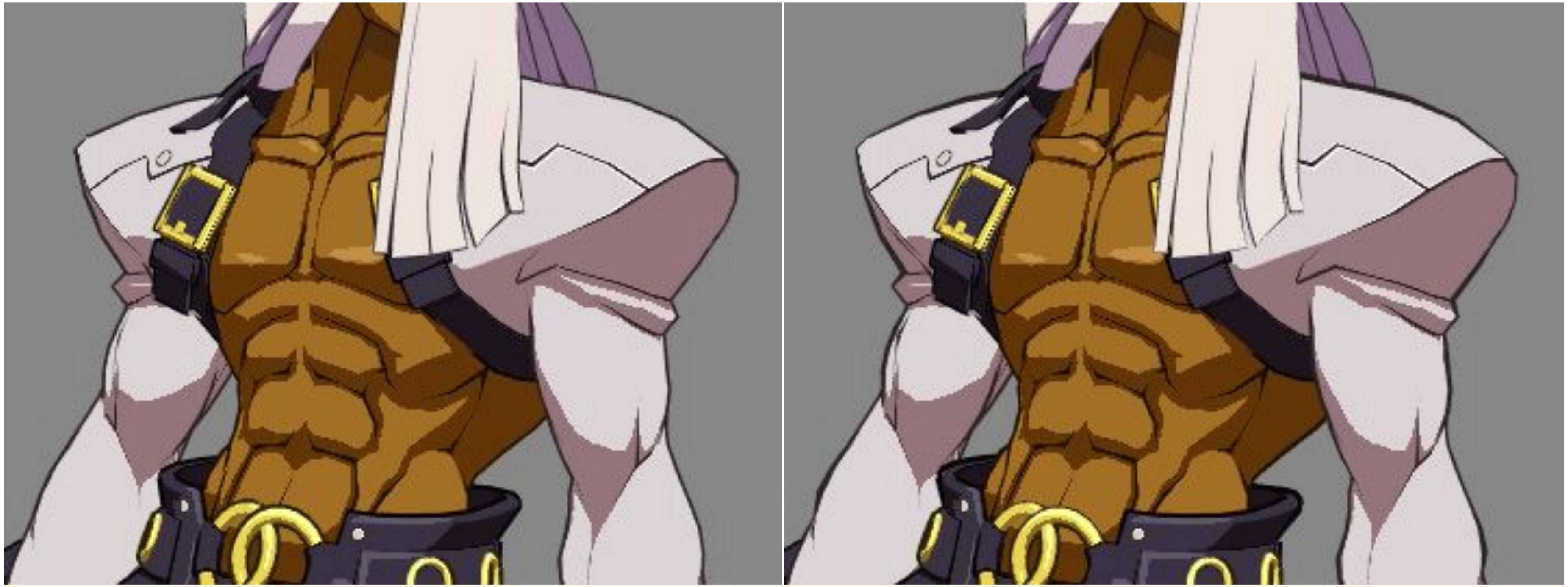
Adjusted line thickness (Displaying the ALPHA Channel of Vertex Color)

The Secrets Behind the Linework



Without (left) and with (right) thickness adjustment. Note the nose, cheeks and chin area.

The Secrets Behind the Linework



Here you can see the changes from the shoulder to arm area. Without (left) and With (right)

The Secrets Behind the Linework

According to Motomura, the breakdown of the use of vertex color in GGXrd is as follows.

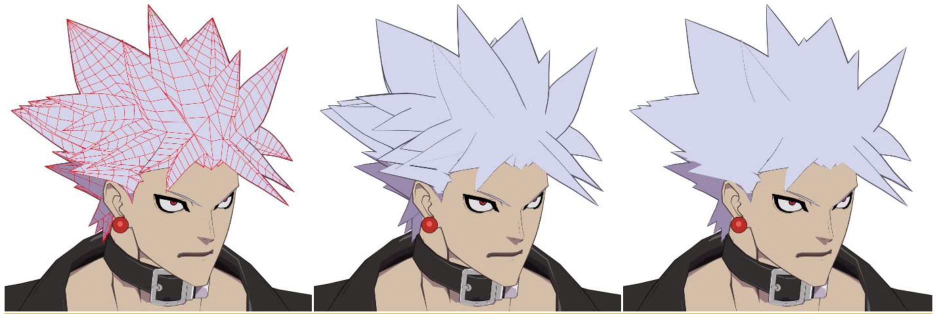
RED: Offset to the shading threshold. 1 is the standard, and the darker the shadowy area, the more likely it is to be shaded. 0 means it will always be shaded.

GREEN: Determines how much the contour line expands compared to distance from the camera.

BLUE: Z-offset value of the contour line.

ALPHA: Thickness factor of the contour line. 0.5 is the standard value, 1 is the maximum thickness, and 0 is no contour line.

Green and **Alpha** are the parameters that **control the thickness** of the contour line; **Blue determines how much the expansion is shifted in the depth direction (Z direction)** relative to the viewpoint. The larger this value is set, the easier it is for the inflated model to be buried in the neighboring surface, resulting in the occlusion of contour lines. Motomura says this parameter was added to prevent unwelcome wrinkle-like lines from appearing in the hair or under the nose.

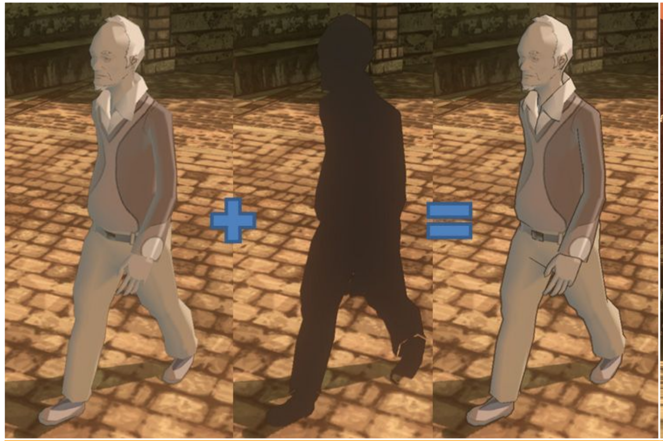

Geometry (left), without (center) and with (right) Z-offsetting the outline.

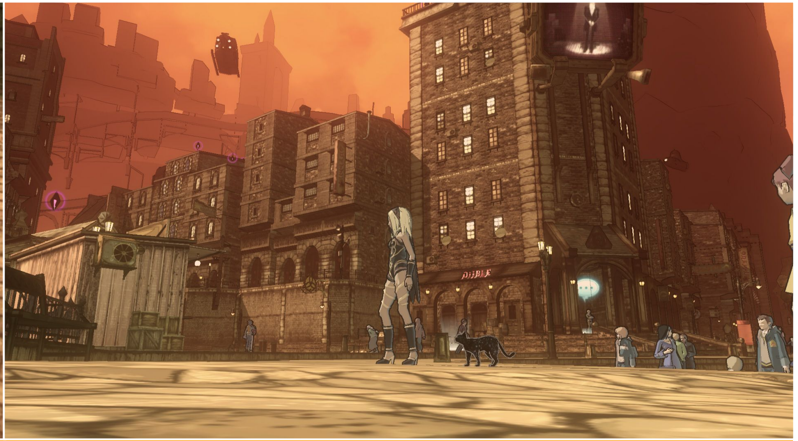
The Secrets Behind the Linework

D. Ishiwatari— *The reason we didn't go for a post-process approach was because we thought **it would be difficult to control the thickness of the lines** as we did.*

With this method, we can adjust how the lines will appear on the final platform, from the 3D model creation stage, on the artist's side, so both the model and outlines are created simultaneously.

The post-processing that Ishiwatari refers to is where lines are drawn on the rendering result, using pixel shaders. Specifically, the contour pixels are determined by detecting the depth difference in the rendering result, or by detecting the difference in the inner product value of the pixel-to-pixel line of sight (view vector) and the direction of the surface (normal vector). This method is often employed when the geometry load is considered to be too high for the inverted hull method, and was recently used by **GRAVITY RUSH** exclusively for the backgrounds.





GRAVITY RUSH (C) 2012 Sony Computer Entertainment Inc.
The character models were drawn using the back method, and the background was drawn using post-effects.

The Secrets Behind the Linework

The newly developed “Motomura-Line”.

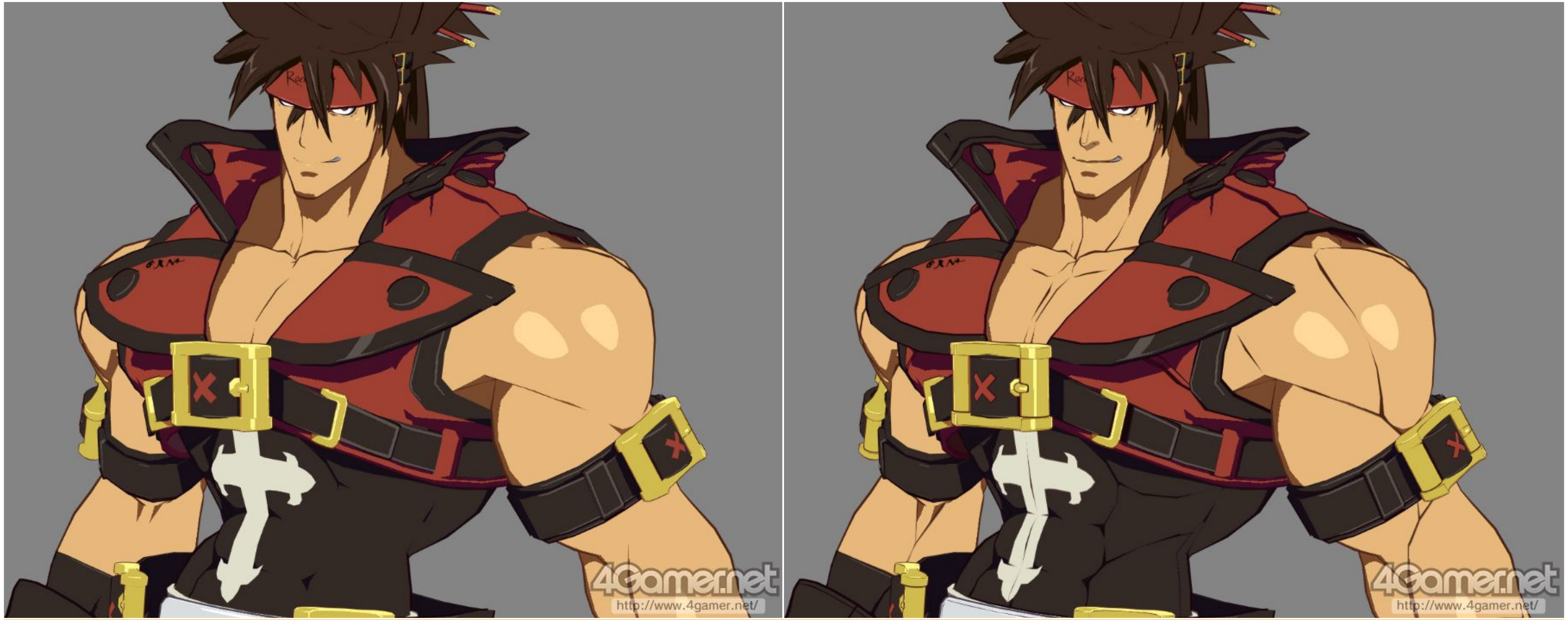
The Secrets Behind the Linework

In GGXrd, they added another method to show contour lines, such as muscle ridges, grooves, seams on clothing and accessories.

J. Motomura— *There are some areas, such as grooves in the 3D structure, where it is impossible to draw outlines using the inverted hull method. And if we use normal texture mapping, jaggies will appear when the camera is zoomed in, and the difference from the clean contours of the back method will be noticeable. Therefore, we went back to basics: "what kind of situation would cause jaggies in texture mapping?" to understand how to create **clean line segments that do not depend on the texture resolution**.*

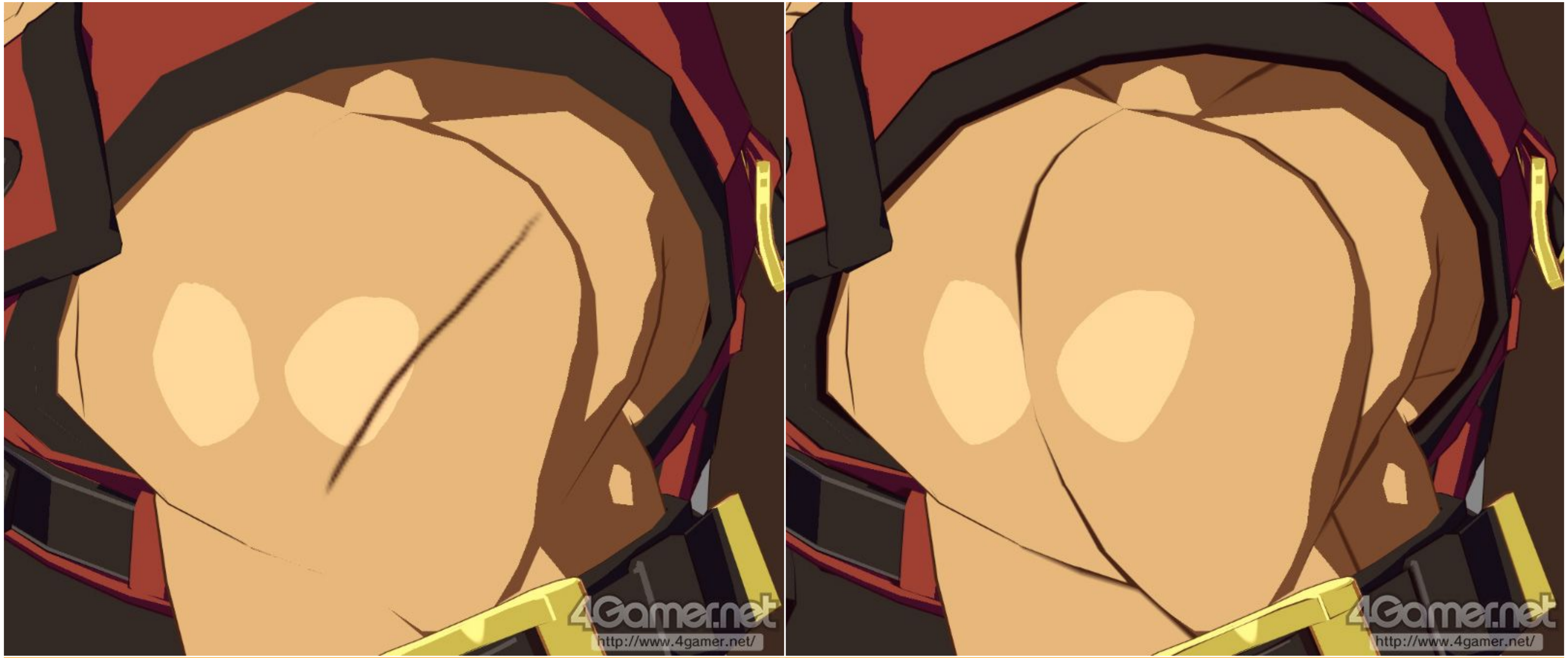
The result was a unique line-drawing technique, called the **Motomura-Style Line** by the development team. →

The Secrets Behind the Linework



The inverted hull outline alone cannot be used to draw a line in any part of the figure. Instead those lines have to be drawn with texture maps.

The Secrets Behind the Linework



Results of drawing a freehand line (left), you can see the pronounced jagged lines. In contrast(right), the "Motomura lines" appear sharp even at the same texture resolution

The Secrets Behind the Linework

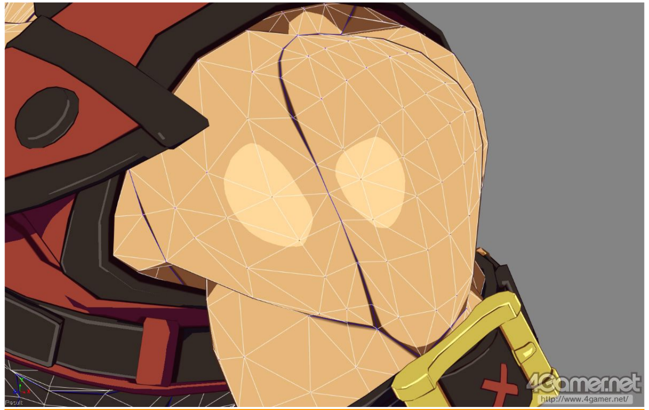
Jagged lines happen in texture mapping when a particular texel (a pixel that constitutes a texture) is drawn as a single texel on a polygon surface. On the other hand, when there are adjacent texels, the jaggies are less apparent because the shape of the square texel virtually disappears.

However, even if they are adjacent to each other, if they are **diagonally above or below each other**, it's basically the same problem as having a single texel, and jagged edges will appear. **In other words, if the pixels are lined up horizontally or vertically, you may get some blur, but the jagged lines can be avoided**.

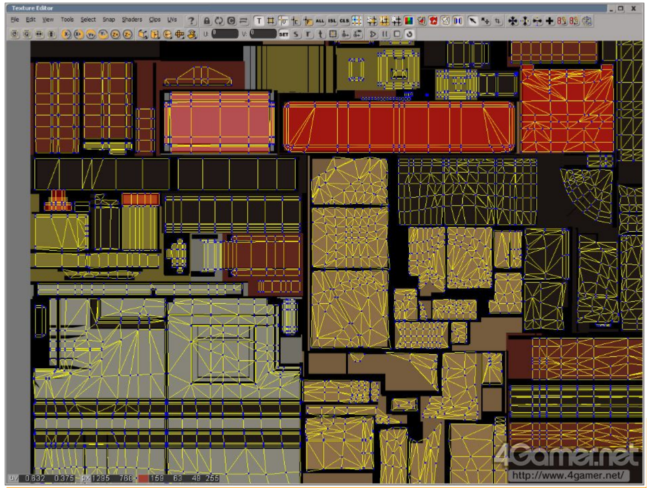
Realizing this, Motomura created a texture consisting of only vertical and horizontal aligned pixels, with solid lines given as contour lines. He then designed a UV map (i.e., data that shows where each polygon on the 3D model corresponds to on the texture map) with distortions and bends so that when applied to the 3D model, diagonal lines and curves would be mapped where desired.

"When I tried texture mapping using this method, I was amazed. Even with a texture of not so high resolution, I was able to obtain beautiful and smooth line drawings."

The actual texture used for the Motomura line, is stored solely within the alpha channel of the ILM texture, since only the black and white information is needed.

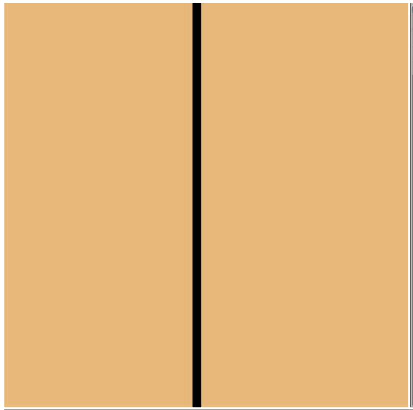


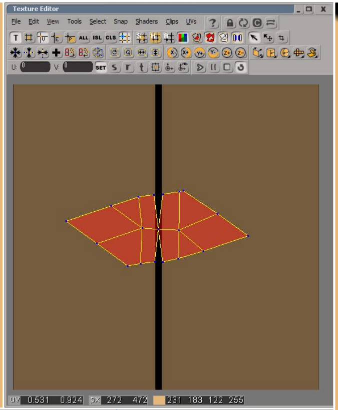
A closer look at thelines (with wireframe)

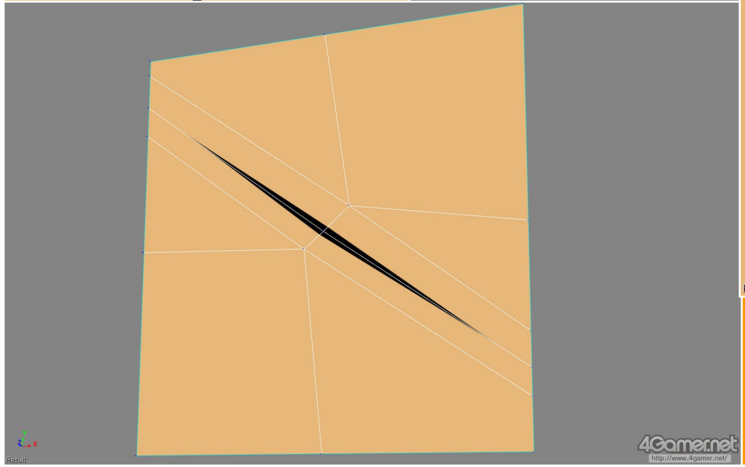


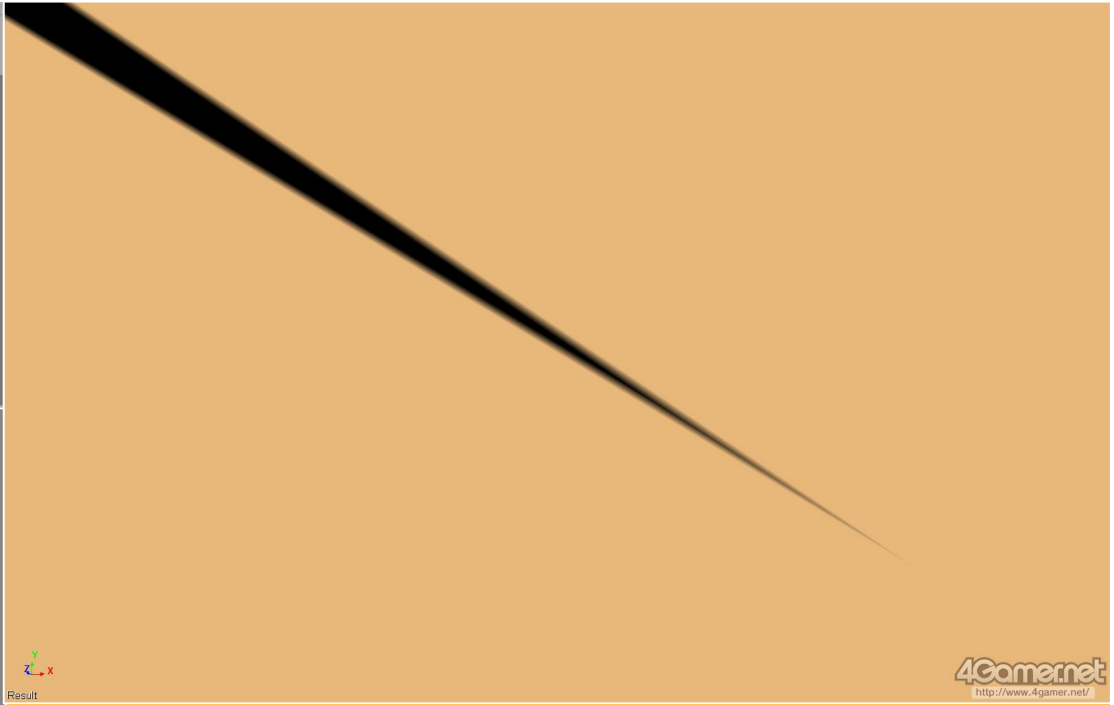
Example of the UV Map for the line technique.

The Secrets Behind the Linework



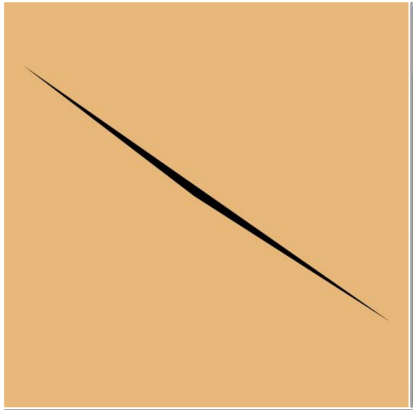

Texture Editor
File Edit View Tools Select Snap Shaders Clips UVs
UV 0.531 0.924 px 272 472 231 183 122 255

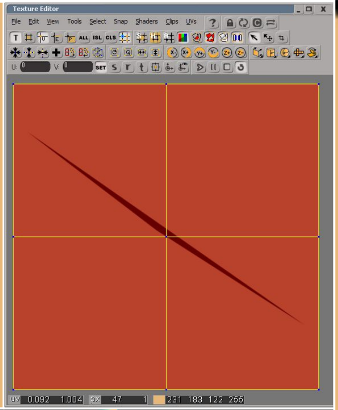


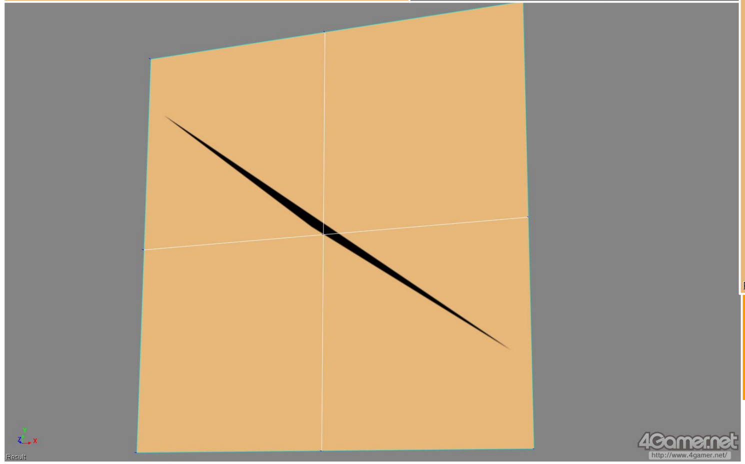


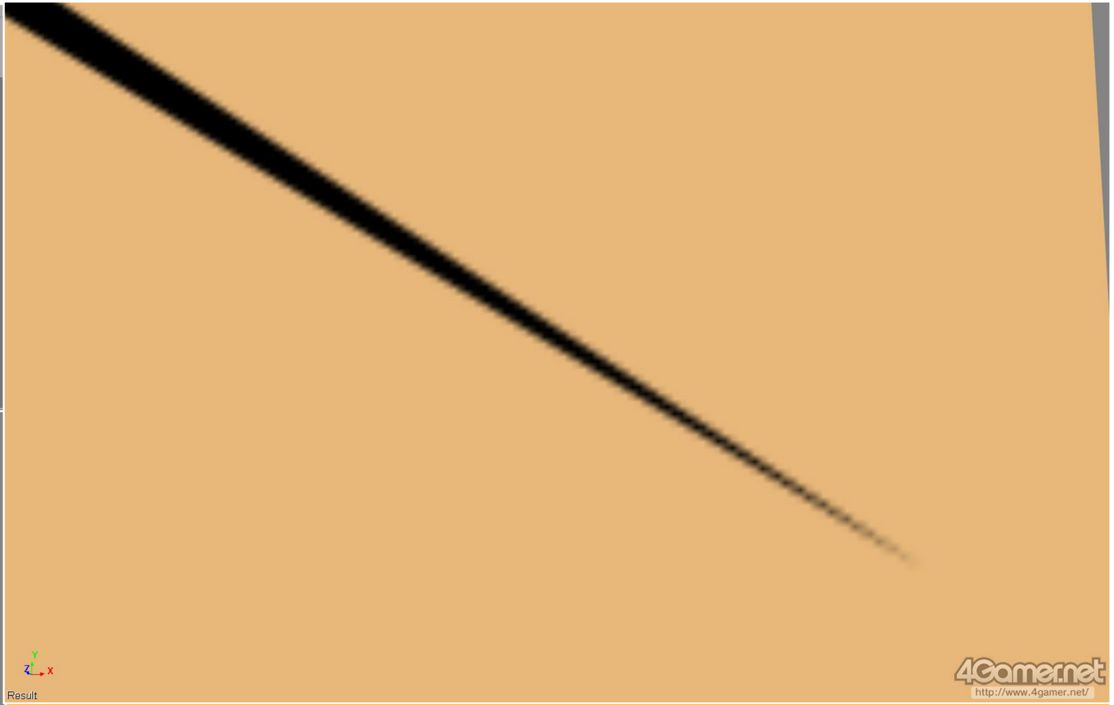
Example of a texture for a Motomura-style line. Here, vertical lines are drawn. (Top Left)
UV unwrap of a Motomura-style line. (Top Center)
The result and an example of the mesh structure. Note that the topology has been devised according to the line to be drawn.(Bottom Left)
An enlarged view of the result of applying the Honmura line. Note the absence of jaggies. (Top Right)

The Secrets Behind the Linework




Texture Editor
File Edit View Tools Select Snap Shaders Clips UVs
U: 0 V: 0 SET
UV 0.092 1.004 px 47 1 231 183 122 255





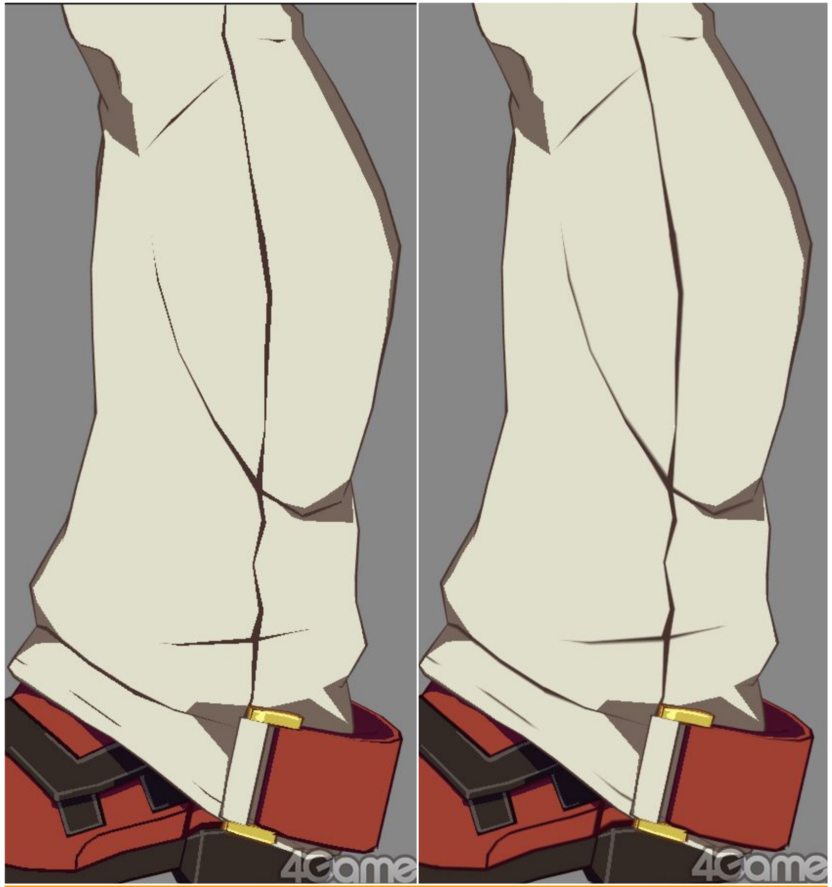
For reference, here is an example of a line texture. You can see that the jaggies are especially noticeable in the final image. The mesh itself is also divided in a way that's completely unrelated todo with the lines to be drawn (bottom left).

The Secrets Behind the Linework

In the example of the Motomura line, you can see its strange texture looks like a city plan, with only orthogonal line segments, but for example, a texture framed by a square line would be an outline applied to a muscle ridge. The muscle ridges have an oval hemispherical shape, but in this method, **that oval is distorted to fill a square shape**.

As the elliptical becomes a square, the area inside it will be **stretched and distorted considerably**. Therefore, if there are any characters or patterns inside the square, they will naturally be distorted as well. However, since this texture is only for adding contour lines, it does not include such symbols or patterns.

Naturally, texture mapping is done based on the distorted and bent UVs, and the texture is affected by scaling and rotation when the character or viewpoint moves. When the character and viewpoint move, the texture is also affected by scaling and rotation, which means that the lines drawn will also be affected by these changes, resulting in curves and slanted lines. But since **bilinear filtering** is applied to the texture mapping, such curves and slanted lines are slightly blurred, which doubles as an anti-aliasing effect.

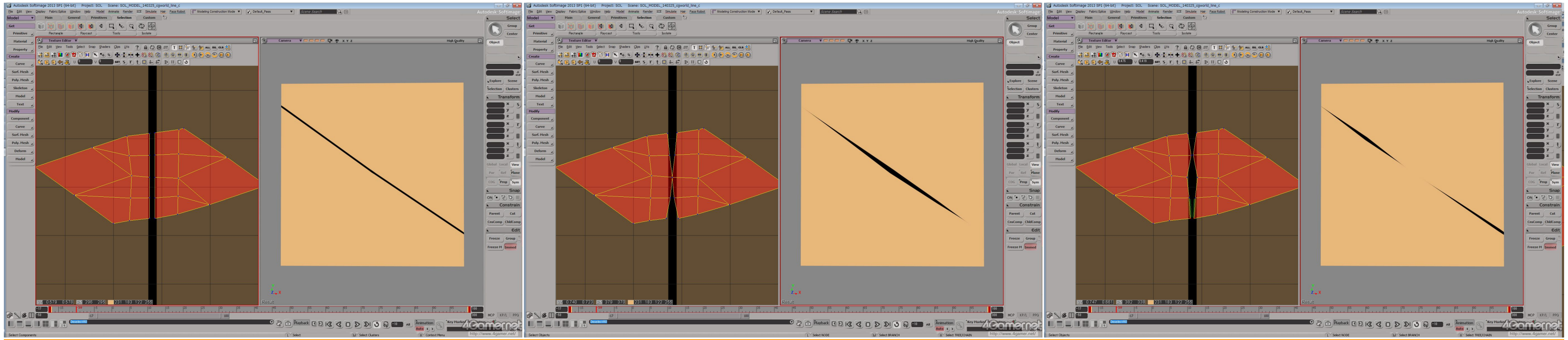


Without (left) and with (right) bilinear filtering.

The Secrets Behind the Linework

By the way, the line segments drawn in the Motomura method also have a thickness variation. However, the texture itself does not have such a subtle line strength, and is basically uniform. So how does the final line have variation?

J. Motomura— *The thickness of the strokes in the Motomura line is controlled by the design of the UV map. If you want to create a thicker line, you can design the UV map so that the polygon surface is more widely allocated to the texels that represent the lines in the texture.*



An example of setting the thickness of the stroked line. The left image is without any settings, and the center image shows how the overlap between the UVs and the line is widened or thinned. As shown on the right, you can also create styles with natural breaks in the lines.

I have a feeling that depending on how you do it, these lines can be applied to a wide range of game graphics other than anime.

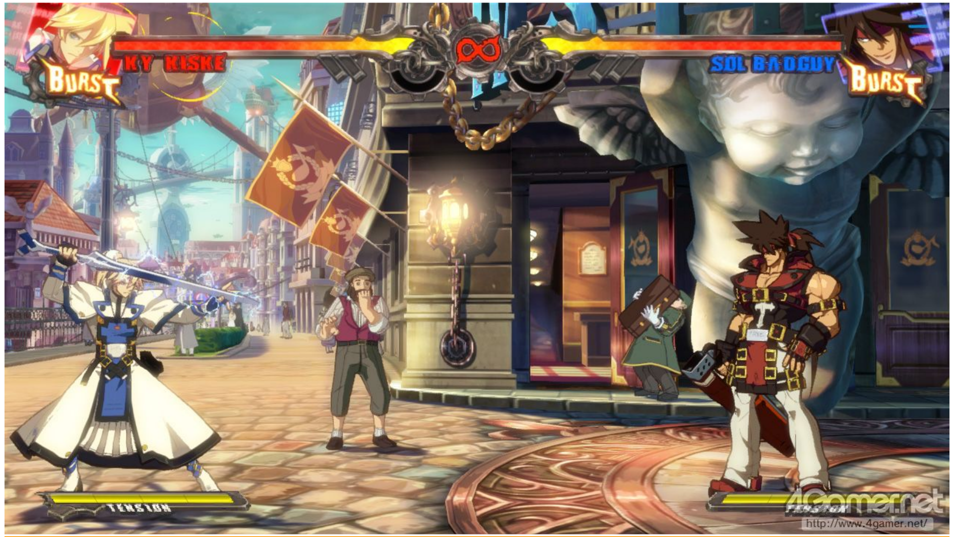
Deformed Geo, Swapped Geo

Deformed Geo, Swapped Geo

The reason why GGXrd's graphics look like celluloid animation is not only because of the use of toon shaders. The unique innovations in other areas of the game also play a major role.

For the battle stages, It's easy to see from our point of view that the various buildings and background objects that line the stage are represented by 3D models rather than "single pictures".

But when the two fighting characters are moved to the left or right, **the near view moves at one speed while the distant view moves much slower**, a familiar visual in 2D fighting game graphics, but in fact there is an incredible secret hidden here.



Ky and Sol fighting each other in Illyria Castle Town. You can see a hotel in the foreground and a street stretching into the distance on the left.

Deformed Geo, Swapped Geo

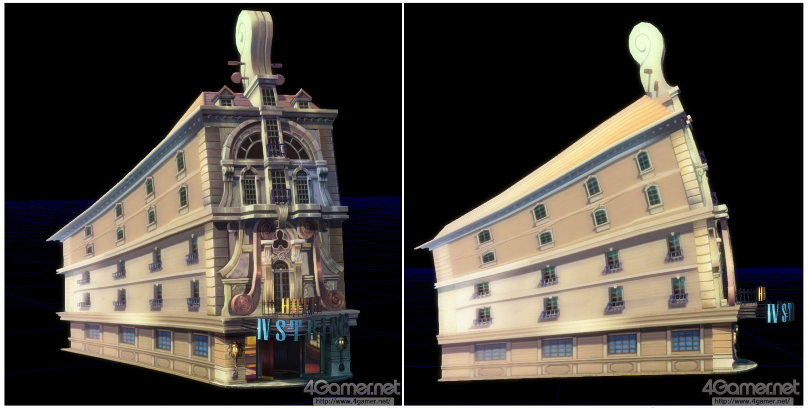
Yes, the background 3D models of the near scenes and buildings in the scene are **modeled with a very strong perspective**, as if they were squeezing into the depths of the scene.

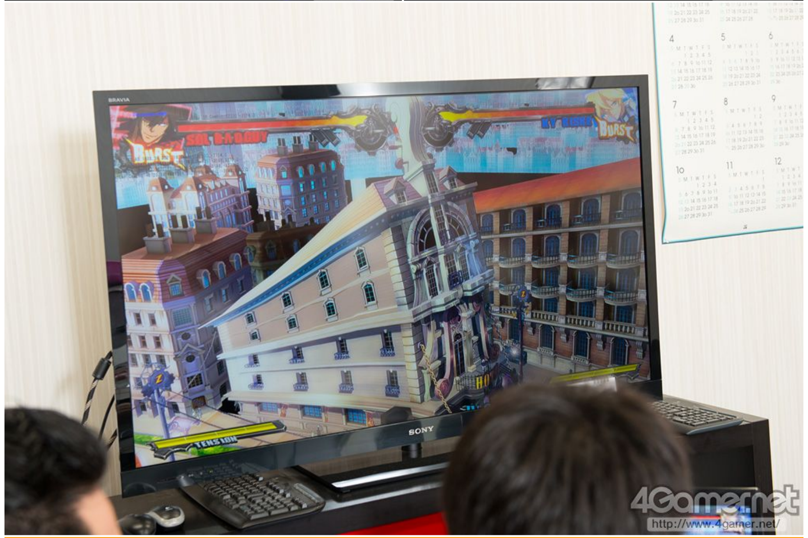
On the other hand, the 3D models of the distant scenes, although not in exact perspective, **are much smaller than the actual scale**. There is a reason for this modeling and arrangement.

Were they modeled in full scale without any perspective, we would not be able to showcase the depth of the scene.

The distant objects are scaled down for the same reason, but they are made smaller and placed not too far away so that they can be moved left and right as the characters fighting in the battle stage move left and right.

In real life, if you wanted a building to appear this small in the composition, you would have to place it at a considerable distance. As you've probably experienced in the real world, the distant view doesn't move much when you move a few meters to the left or right. **This would mean that the background would not move as it does in 2D fighting games**, so instead, small 3D models were placed in close proximity to have them move as well.

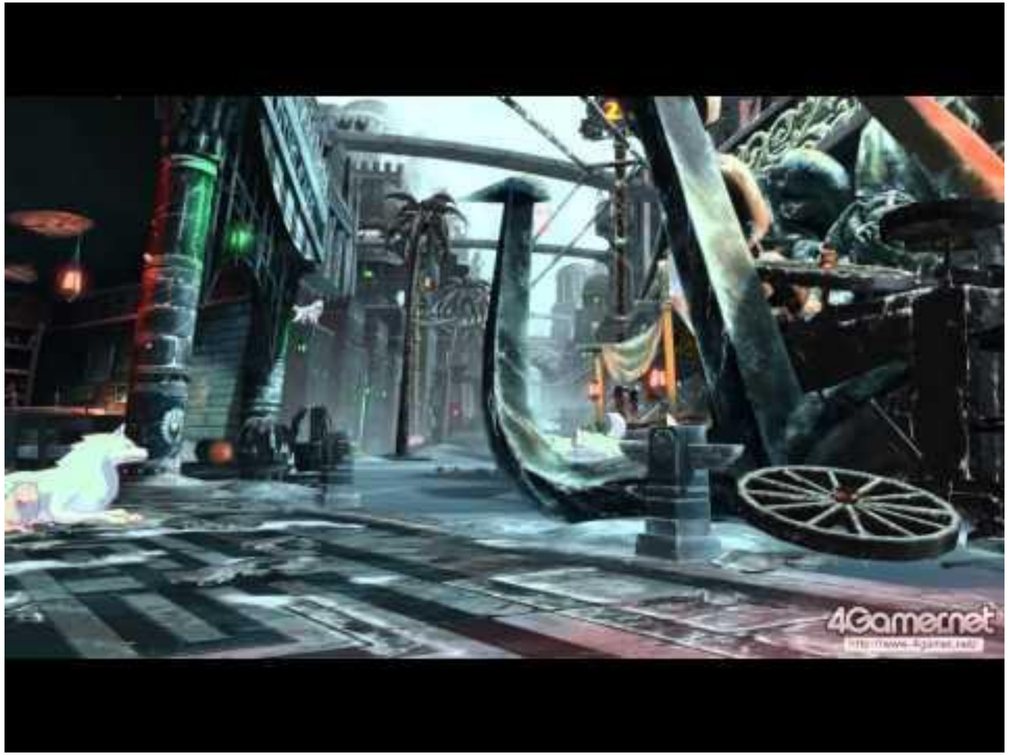




The Illyria Castle Town stage, a mesh of buildings is displayed by itself, and the camera is moved. The graphics are quite surprising.

Deformed Geo, Swapped Geo

Here’s an example of the movement in a battle stage.

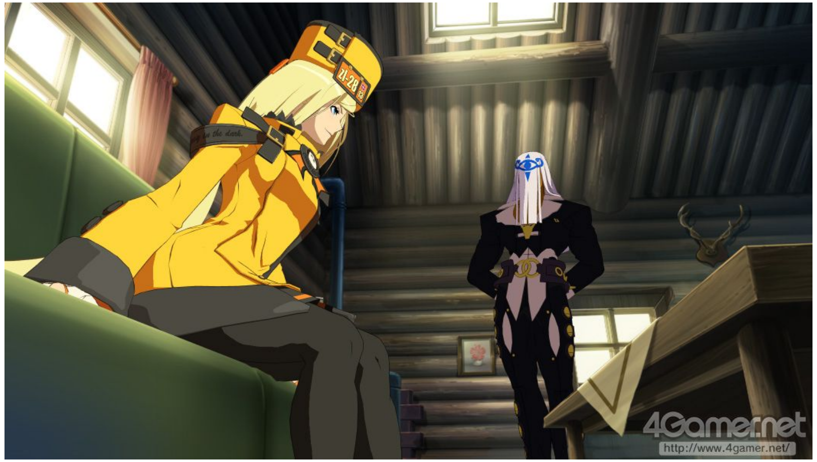


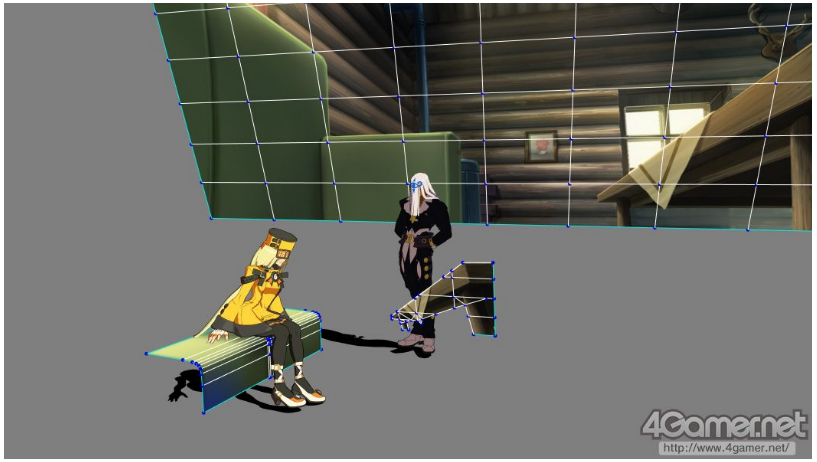
Deformed Geo, Swapped Geo

After reading the above explanation, some of you may be wondering what is going on in the game world behind the camera. Ishiwatari explains:

D. Ishiwatari— *In the K.O. scene, the camera goes around and faces the front side of the screen. So we set up background objects in the foreground as well, as far as the camera can see.*

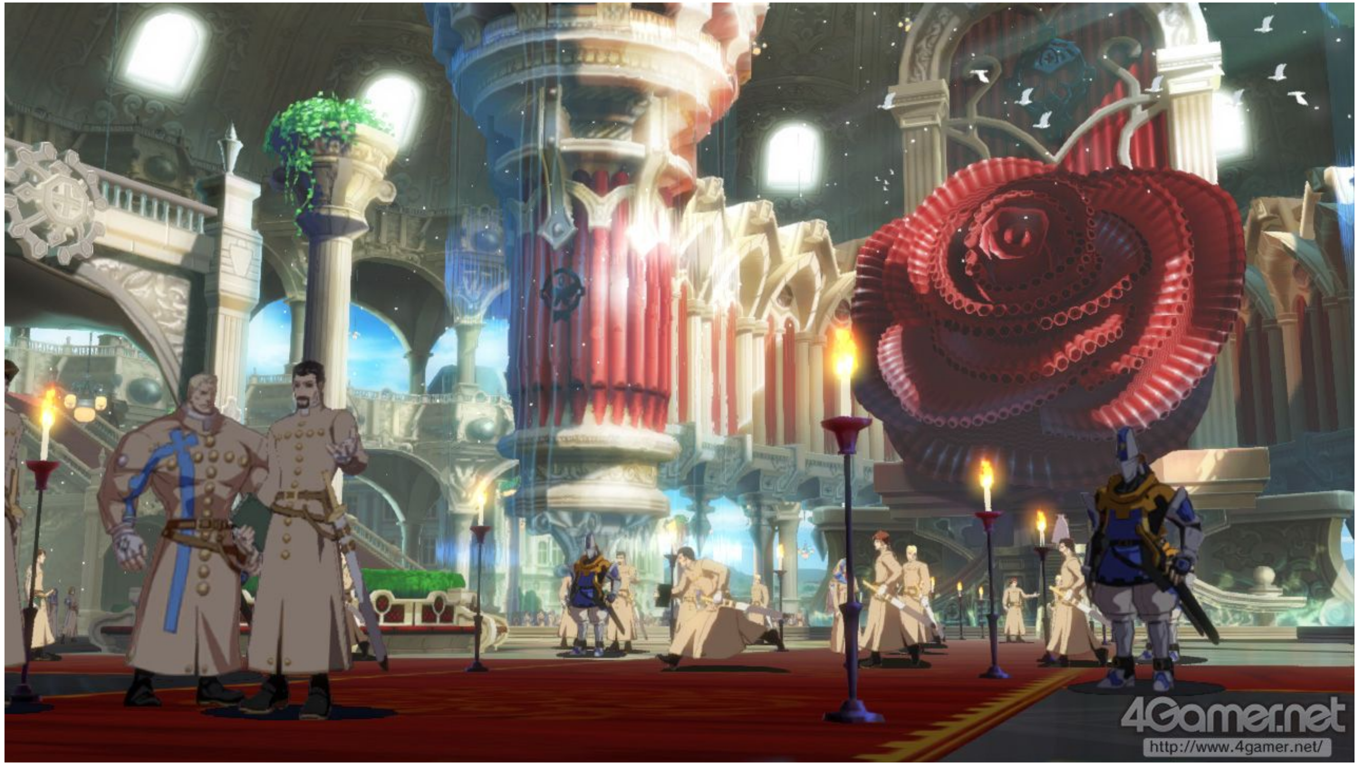
Also, in some cutscenes, we use 2D drawn backgrounds. When we want to move the 2D background dynamically according to the camera work, we use a TV animation-like theory for the 2D background itself (laughs).





Deformed Geo, Swapped Geo

One additional note on the backgrounds: the mob characters close to the battle stage are 3D models, while the ones in the distance are animated billboards in a *PaRappa the Rapper* style.

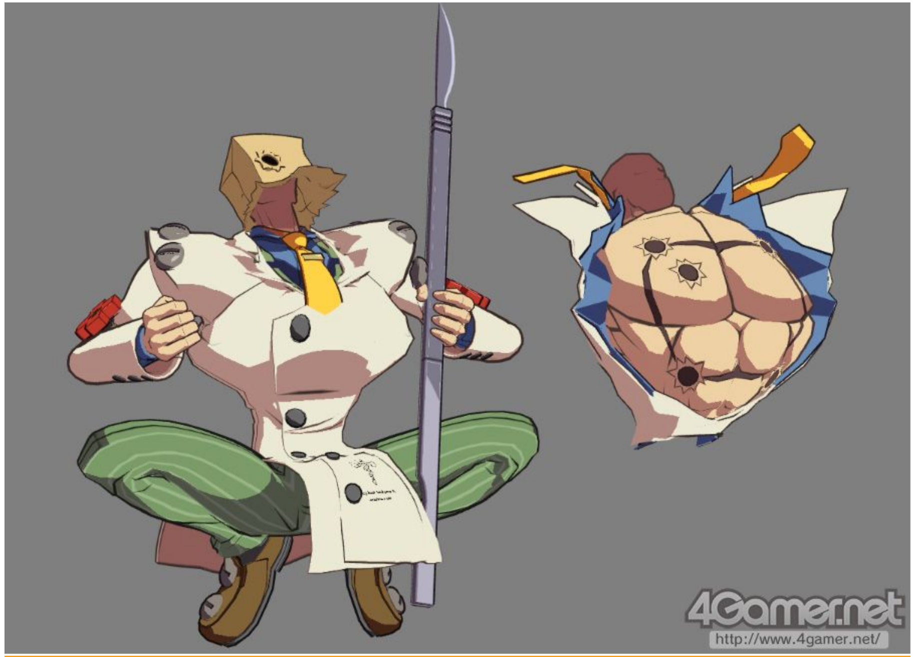


Deformed Geo, Swapped Geo

As mentioned before, the main characters in the battles are modeled with about 40,000 polygons, but they are animated in a slightly different way than the usual 3D graphics-based game characters.

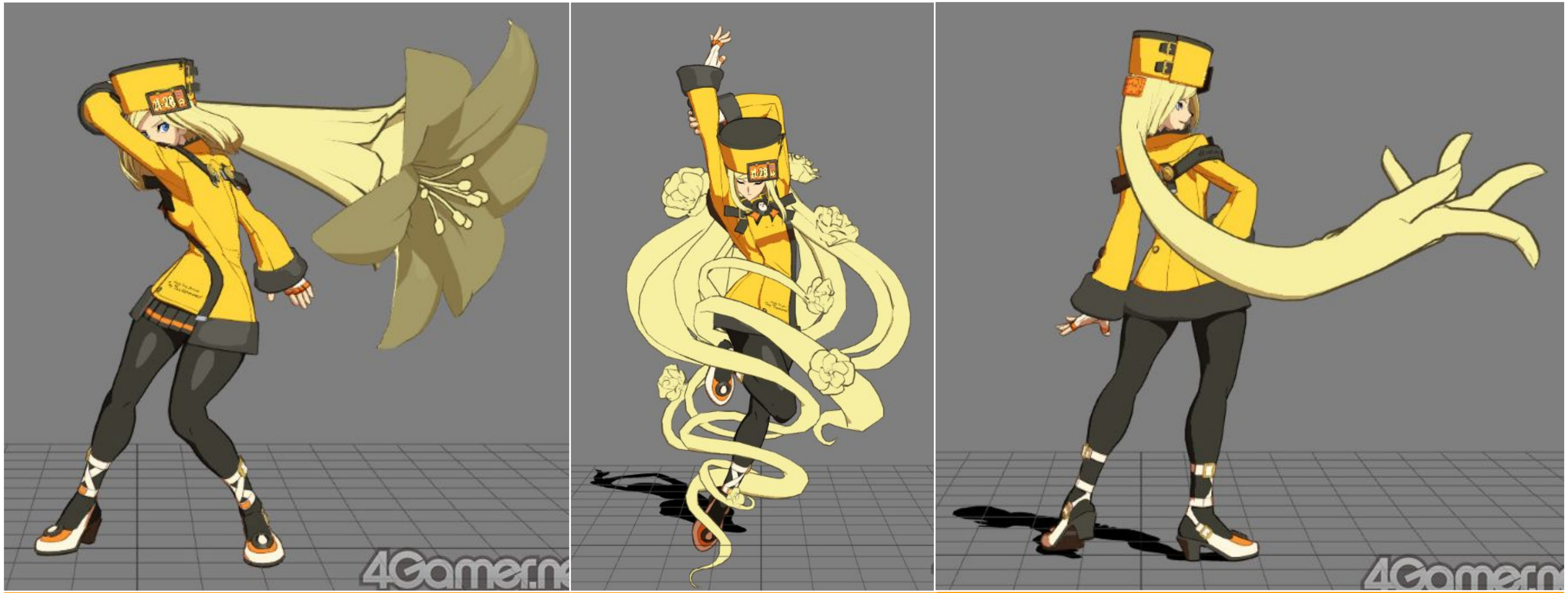
In general 3D graphics, game characters are posed, deformed, and animated by moving the bones inside the model and having the outer 3D model follow suit.

In GGXrd, most actions are implemented in this way, but this is not the case for attack actions where hair transforms into the shape of a blade, for example. So in such special cases, **they deal with it by replacing some parts of the 3D model**.



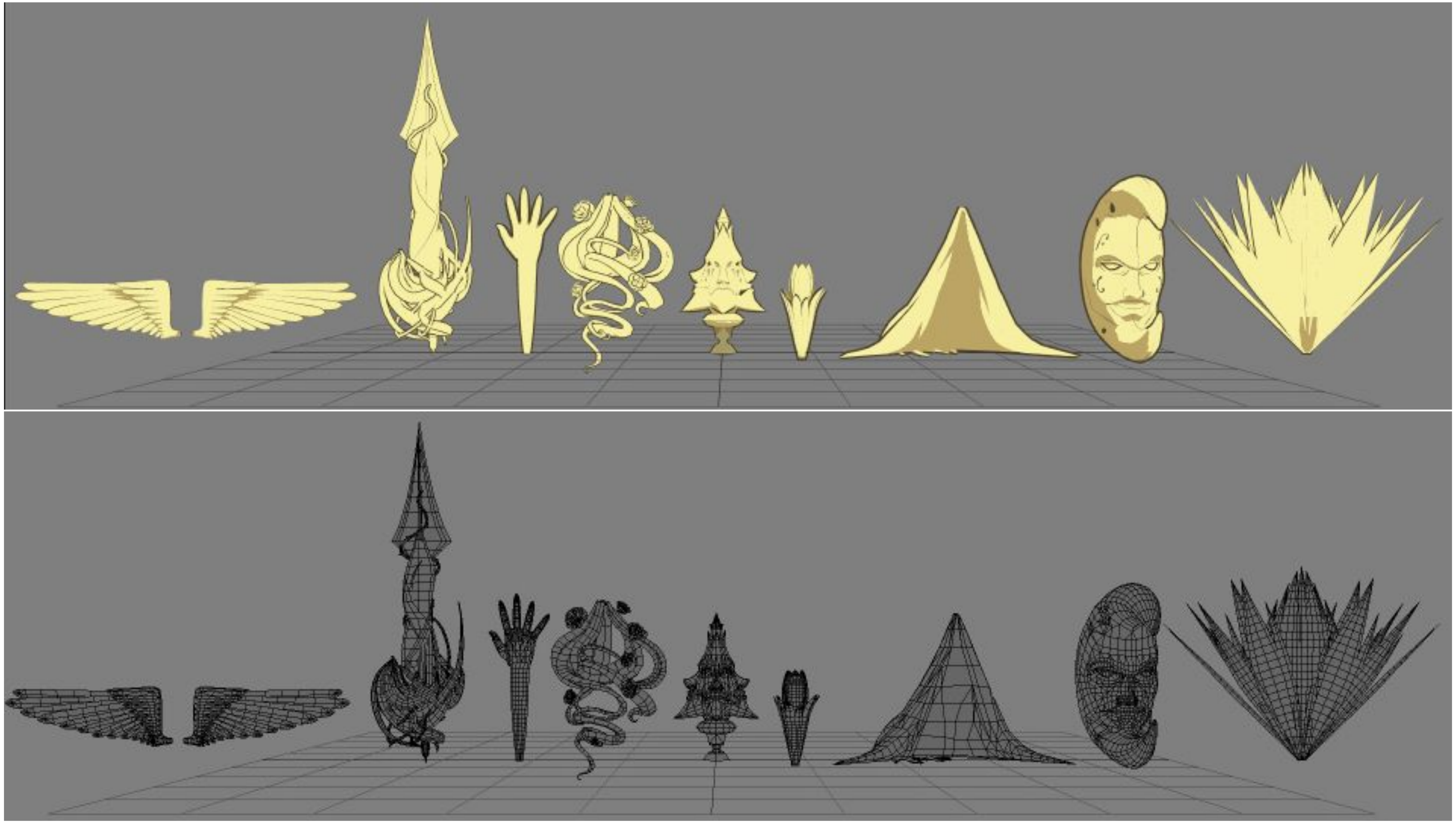
Faust's opened shirt is made with additional parts.

Deformed Geo, Swapped Geo



In the case of Milia, whose hair is transformable, additional parts are made, which can be replaced.

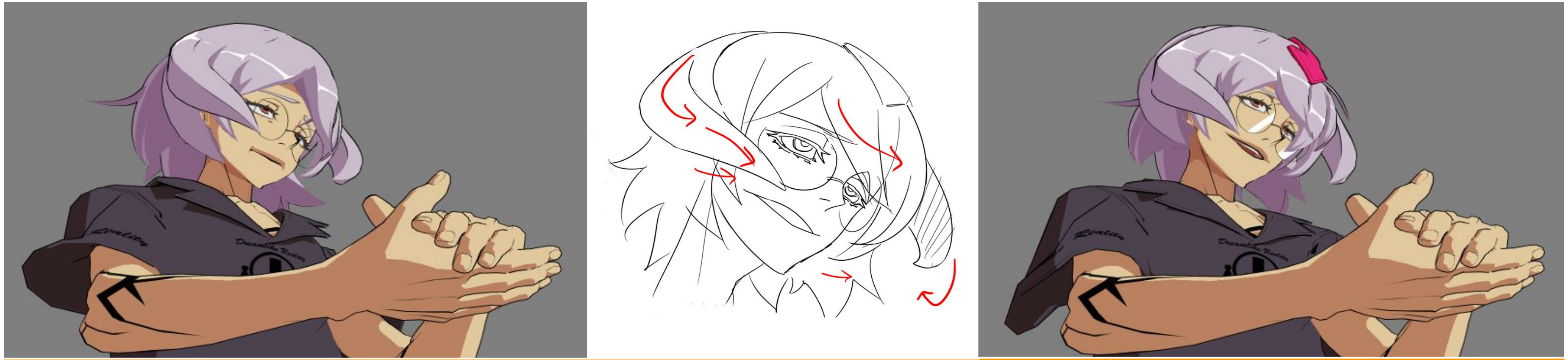
Deformed Geo, Swapped Geo



Deformed Geo, Swapped Geo

H. Sakamura— *Each character's movement is controlled in a unique way to create a dynamic, hand-animated style. Unlike most 3D game graphics, GGXrd characters' eyes, noses, and mouths on their faces "move" a lot. In addition, parts of the body that would be impossible in a real person will expand, contract, or shrink.*

It's all about looking good on the game screen for the player. *This is the motto of GGXrd's graphics.*



An example of the use in Bedman's Instant Kill cutscene. If you create facial expressions in a conventional way so that they look safe from any direction (left), you won't capture same expression as the reference (center). To solve this problem, the position and angle of the eyes, nose, and mouth are adjusted using a rig (right), and the character is effectively presented by targeting only the image as seen by the camera.

Deformed Geo, Swapped Geo

The characters are equipped with bones same as other 3D games, but **are given a high degree of freedom in how they are moved and controlled**.

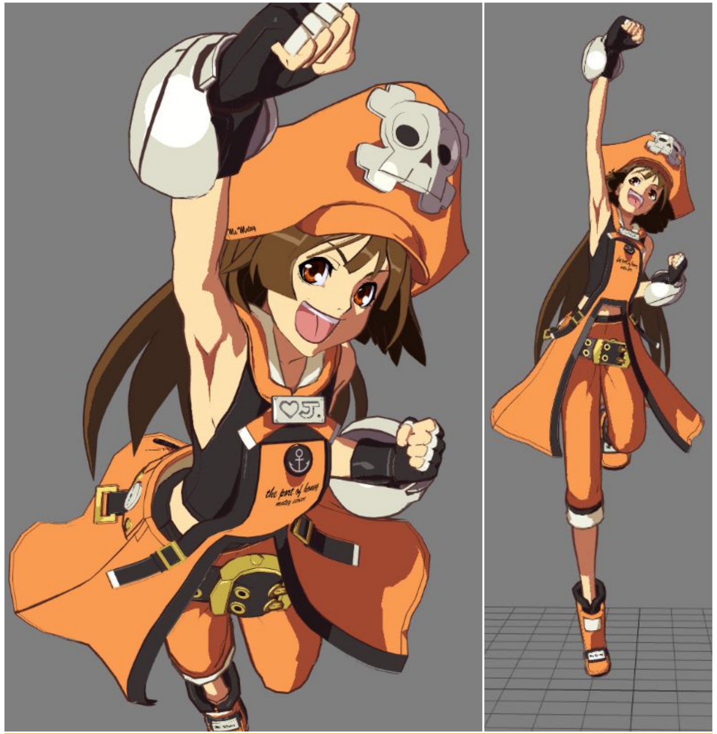
For example, the eyeballs can move freely within the face, and the mouth can deform and open and close to the extent that the volume of the face changes considerable. The arms and legs can grow and shrink freely. This is implemented to achieve the "cartoony and aggressive facial visuals" that we are used to.

However this system cannot handle cartoonish expressions such as comical "round eyeballs" or anger marks. Therefore, when such expressions are necessary, they are handled by replacing parts, adding parts, or using textures.

D. Ishiwatari— *Some of the poses, such as reaching out arms and legs, could not be expressed dynamically if we only adjusted the angle of view.*

If we widened the angle of view to emphasize the perspective to make a reaching hand look bigger, the face in the back would become too small.

Instead, we would extend the arm of the 3D model and bring the hand closer to the camera. In the end, we didn't change the angle of view of the camera, but rather we added the angle of view to the character models (laughs).


Final shot and revealing the deformations to achieve it

Deformed Geo, Swapped Geo

H. Sakamura— *Actually, **this kind of scaling and stretching of bones for each part of the character model was not possible with the standard UE3.***

T. Ieyumi— *In a fighting game, it's not uncommon to emphasize parts or postures depending on the action. In a punching action, the fists are slightly enlarged, and in a kicking action, the legs are sometimes extended. UE3 did not support the x-, y-, and z-axis individual scaling systems for bones that are necessary for those visuals.*

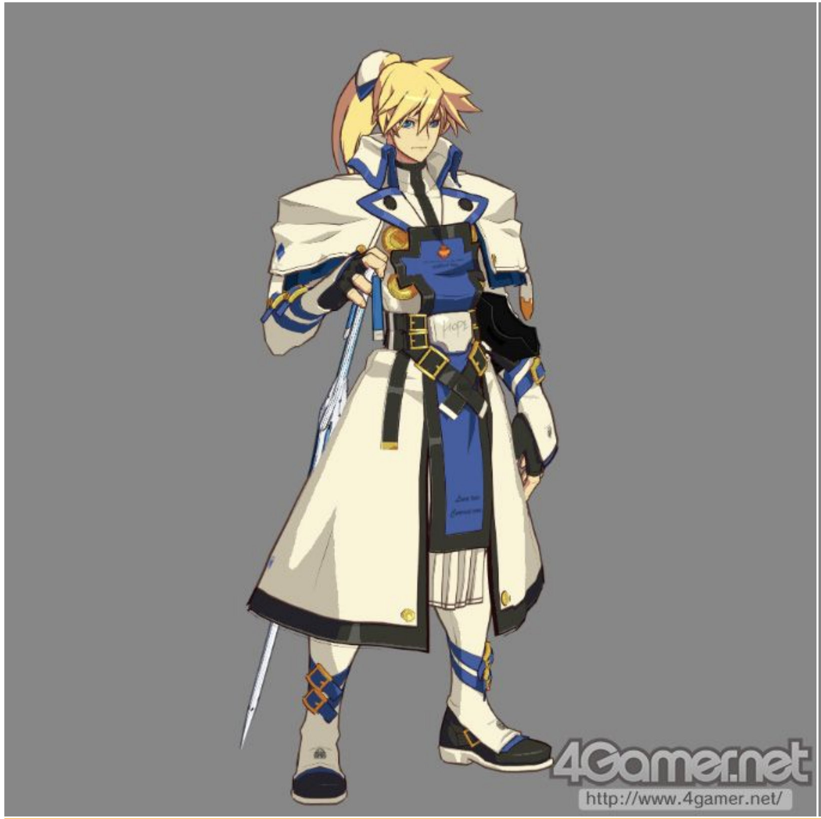
T. Yamanaka— *UE3 is a game engine that was originally designed for photorealistic game production. It may be that the unrealistic "freely scaling bones in three axes" wasn't needed.*

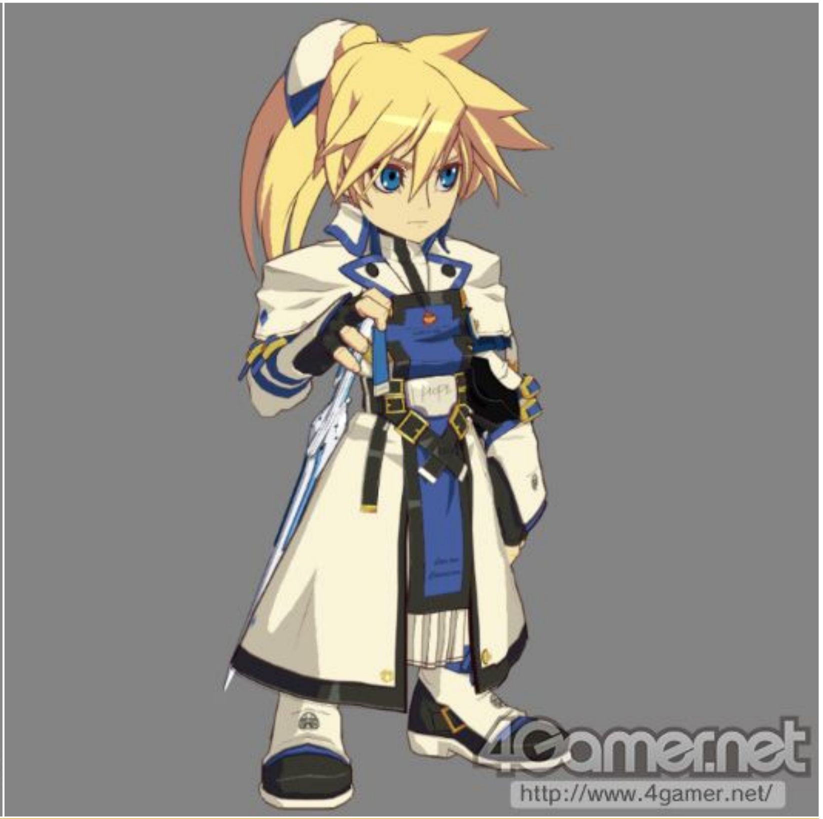
J. Motomura— *In the standard state of UE3, it was possible to scale bones by applying a single parameter to all three axes simultaneously. Using this method, we could create a "super deformed" (chibi) character, though.*

*"In order to create effective animations, **a system that allows the bones to be scaled up and down in each of the three axes is absolutely necessary**"* insisted Sakamura, the animation supervisor. The lead programmer, Ieyumi, half-heartedly agreed and modified UE3 to implement this system. He recalled, *"We were able to do this because UE3 provided the source code."*

Initially, Ieyumi suggested to Sakamura that it might be possible to deal with the problem by creating more bones and moving the bones themselves, but Mr. Sakamura objected. →

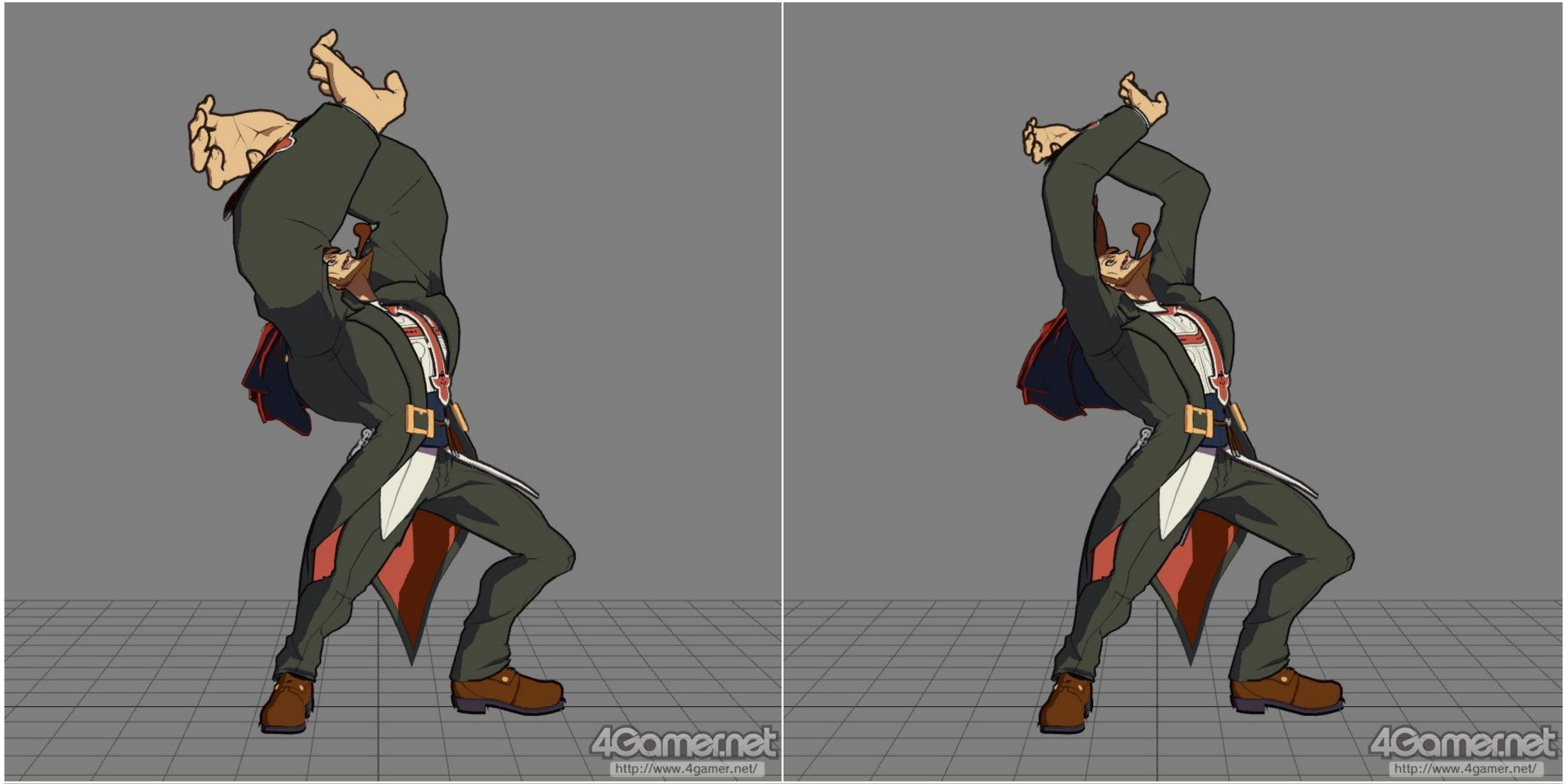
Deformed Geo, Swapped Geo





Normal-scale and “super-deformed” characters. Note that the deformed character shown here (right) is not the result of the basic UE3 function of simultaneously scaling up and down three axes, as described in the text, but of the changes Arc System Works implemented into the engine.

Deformed Geo, Swapped Geo

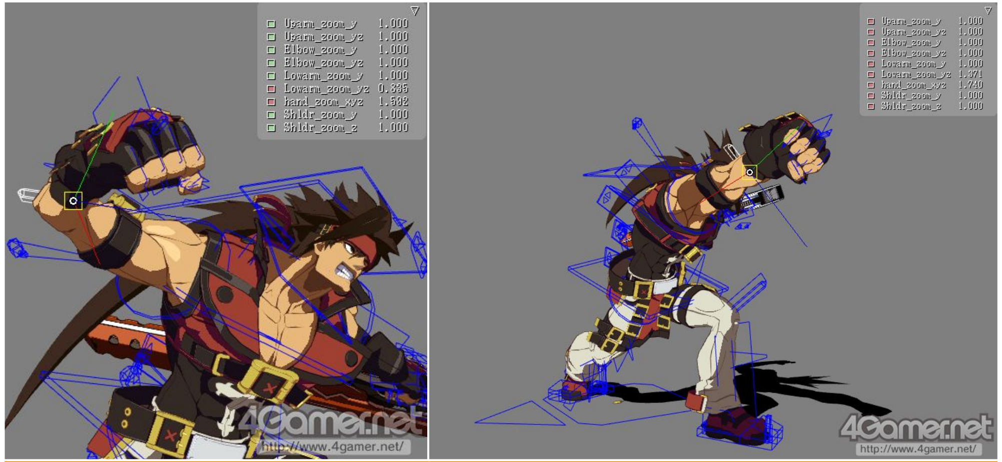


An actual use cast of bone scaling. On the left is the final animation, and on the right, the animation without the scaling of bones.

Deformed Geo, Swapped Geo

H. Sakamura— *It's true that scaling of monster parts was implemented in some unmodified UE3 titles. I guess that was acceptable because they were just some characters. But for us, **all the characters needed to have this level of expression**, and moreover, when we calculated the number of bones we would need for all the limbs, we found **we'd need 4 times as many**.*

T. Ieyumi— *I had to break when I found out about that (laughs). Quadrupling the number of bones would be really hard from a performance standpoint.*



```
Uparm_zoom_y     1.000
Uparm_zoom_yz    1.000
Elbow_zoom_y     1.000
Elbow_zoom_yz    1.000
Lowarm_zoom_y    1.000
Lowarm_zoom_yz   0.835
hand_zoom_xyz    1.592
Shldr_zoom_y     1.000
Shldr_zoom_z     1.000
```

```
Uparm_zoom_y     1.000
Uparm_zoom_yz    1.000
Elbow_zoom_y     1.000
Elbow_zoom_yz    1.000
Lowarm_zoom_y    1.000
Lowarm_zoom_yz   1.371
hand_zoom_xyz    1.740
Shldr_zoom_y     1.000
Shldr_zoom_z     1.000
```

An example of limb scaling in the viewport. The rig is set up so that the user can make changes by simply moving numbers up and down in a small window in SoftImage.

As a result, Ieyumi modified UE3 to support the three-axis individual scaling of bones. Before moving on to the actual production stage, he also created a custom interface that allows the user to easily scale, expand, and contract limbs. Thus, they were able to create animations while using this custom interface to adjust the scaling and expansion of limbs.

T. Ieyumi— *We went through a lot of trouble to get it working, but it was implemented as a standard feature in UE4, which came out afterwards. Well, I guess a lot of people other than us were requesting it (laughs).*

3D Modeled Effects

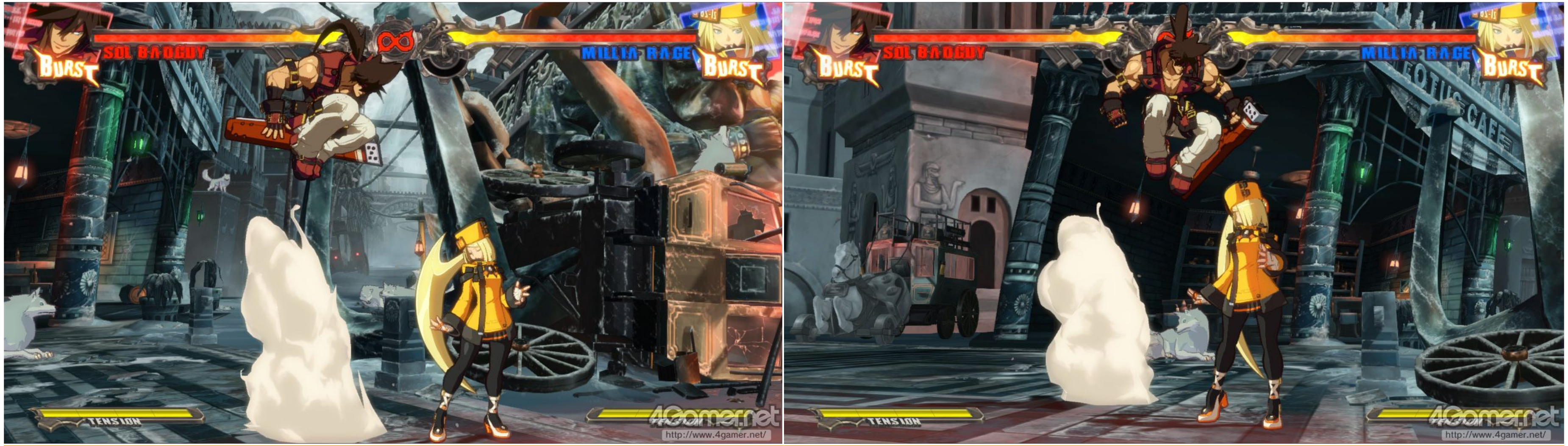
3D Modeled Effects

With a few exceptions, many of the smoke, flames and various flashing effects that accompany the special moves you perform are **also modeled in 3D**.

D. Ishiwatari— I felt it would look awkward if the effects were billboarded and flimsy, so we took the liberty of modeling the effects in 3D (laughs).

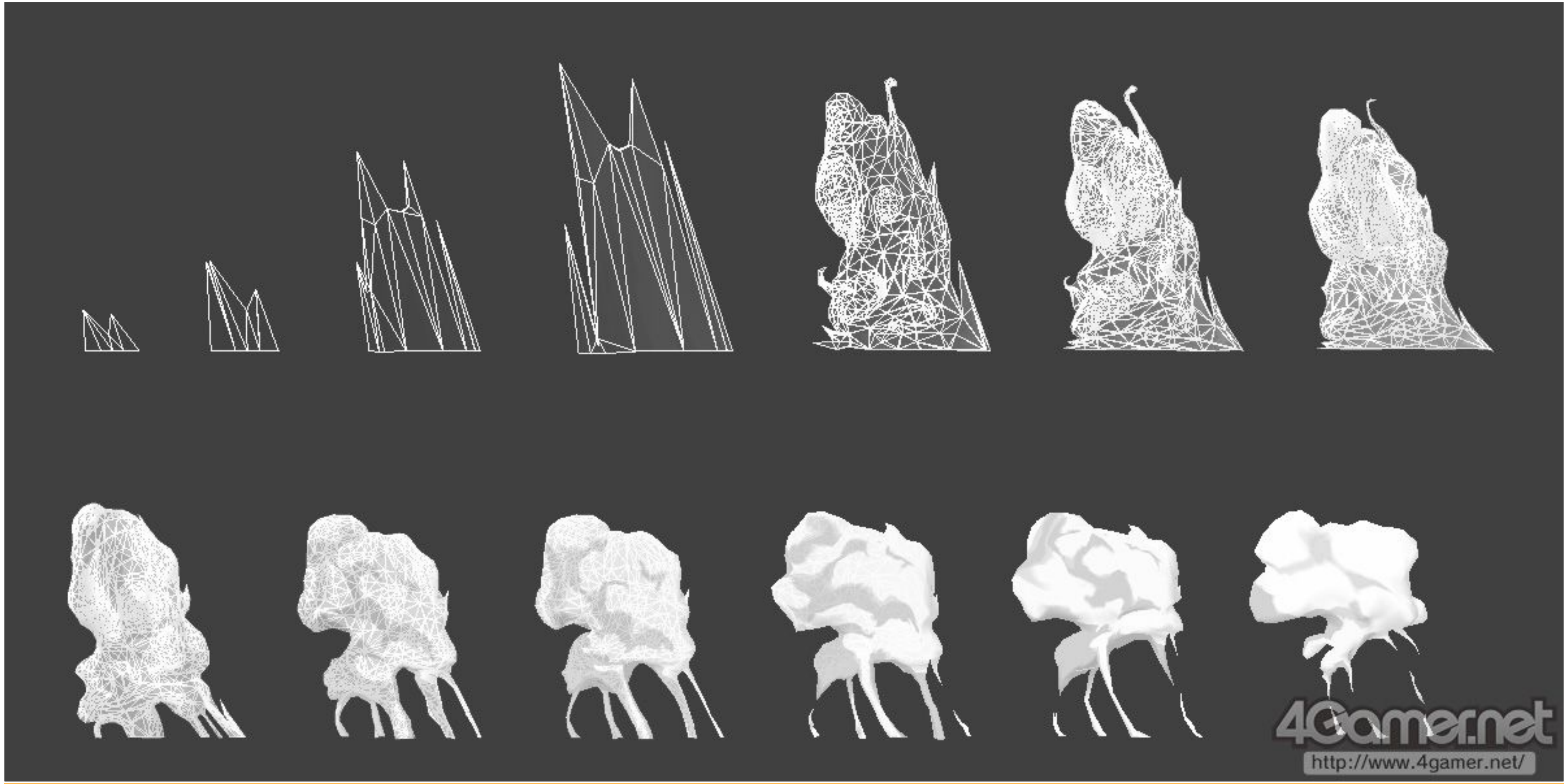
This is easier said than done, but it was skillfully implemented.

For example, the smoke effect shown below looks like a bit of fluid animation, but in fact, it was modeled frame by frame by the artist. The smoke effect, which looks like it is rising, expanding, and bursting, was also created frame-by-frame by the artist to look like that.



Smoke effect (left). Since it is modeled 3D data, the appearance changes when the camera is moved (right).

3D Modeled Effects

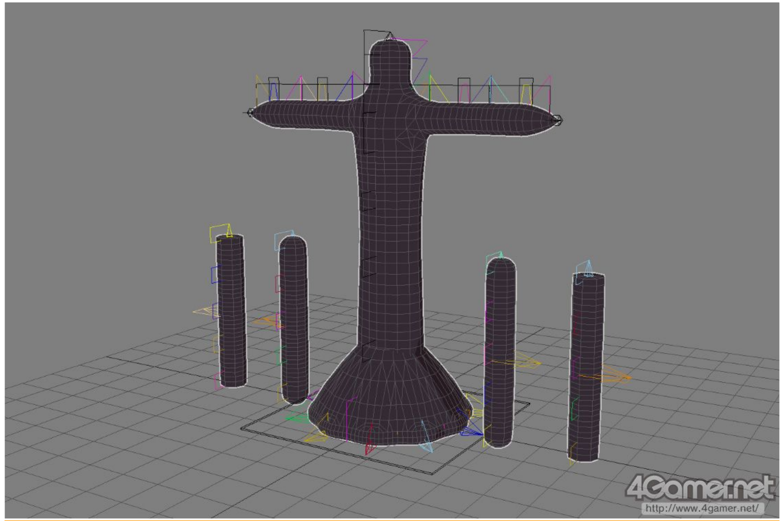


Wireframe and texturing of the model.

3D Modeled Effects

H. Sakamura— *Initially, we experimented with moving multiple sphere models connected by bones, but it didn't turn out to be satisfactory. ...... In the end, I had to resort to forceful techniques (laughs).*

In the case of Zato, a character who manipulates shadows, some of his moves involve melting into shadows, so I transformed Zato's character model up to the middle of the move and then swap it with a "shadow melting" effect model that I prepared separately.

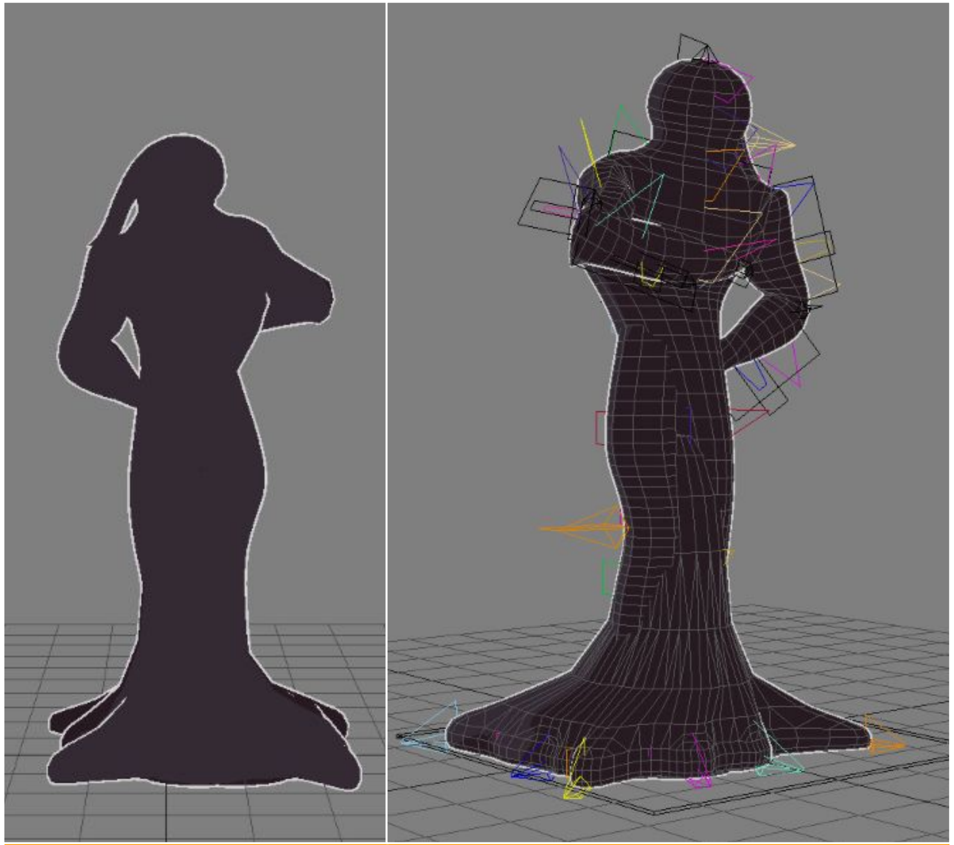


Zato's shadow melting model, before animation.

The smoke effects we talked about are not dynamically lit, their shadows are baked in via texture, and have no outlines. Luminous effects such as flames and flashes **only appear to glow**, but do not actually illuminate the surrounding area.

On the other hand, lighting is used for effects that are part of the character, such as the Zato's shadow-melting, and the company will carefully consider the use of 3D effects in the future.

3D Modeled Effects



The final model on the left, after animation, and as seen from a different angle on the right, proving it is 3D

Creating Cel-like Limited Animations

Creating Cel-like Limited Animations

"Cel shading" has become quite known among gamers. Despite this, many people who see GGXrd for the first time can't tell that the graphics are 3D-based. One of the reasons for this is that **the game's graphics don't move smoothly**.

There have been many cel shaded game graphics in the past, but since they were 3D graphics, they moved smoothly at 30 or 60 frames per second. This was the case with **Jet Set Radio** (SEGA, 2000) for the Dreamcast, and **XIII** (Ubisoft Entertainment, 2003), mentioned in the first part. Naturally, after all, they are based on real-time 3D graphics, so every frame is generated when the character moves.

However, Arc System Works decided that this smoothness didn't suit the feeling of Guilty Gear.

As some of you may know, cell-based animation, such as TV anime, is made up of 8 or 12 frames per second. Therefore, when you are watching TV anime and encounter a scene where drawn characters and CG-based mecha live together, you may have noticed the difference in smoothness of movement between characters moving at 8 frames per second and mecha moving at 60 frames per second(*), and it may have felt weird. Perhaps people who are used to watching traditional animation are more uncomfortable with the smooth movement of anime-style designs.

TN.: The author makes a mistake here. TV animation runs at approximately 24 frames per second, so a mecha in anime would never appear to be 60 frames per second. Secondly, although he is correct that it is mostly animated at 8 or 12 frames, it is not too rare to see more intense or important movement animated at full 24 drawings per second.

Creating Cel-like Limited Animations

H. Sakamura— ***In the end, we went with a 3D graphics system that can animate at 60 frames per second, but we dared to use a frame rate that was more like drawn animation.*** *15 frames per second is the basic rate for Guilty Gear’s cutscenes, which is slightly higher than the 8-12 frames per second of cell animation.*

*Since the battle animation is directly related to the performance of the attacks, the number of frames to display for each pose is specified separately for each attack. The amount of frames each pose is displayed is not fixed, but is rather something like "2F, 3F, 3F, 1F, 1F, 2F, 2F, 3F, 4F" (*Author's note: 1F≈16.67ms) for each technique.*

In the animation industry, "full animation" is when all 24 frames per second are drawn and animated, as in the case of Disney(*), while "limited animation" is when the same frames are displayed in succession and moved at 8 to 12 frames per second.

This animation style is quite different from that of normal 3D game graphics.

In general 3D graphics, character animation is based on creating "trajectories" of bones inside a character model. The trajectory is created by using a high-degree curve function (F-curve) or based on data obtained from motion capture.

On the other hand, GGXrd's limited animation is done by moving the characters little by little and creating poses frame by frame. In other words, **it's more like stop-motion**.

H. Sakamura— *In the beginning, we tried using F-curves to move the characters at 60 full frames per second, and then just reducing the frame rate. But that just looked like 3D graphics with dropped frames (laughs).*

TN.: Here the author repeats a misconception about Disney and full animation. Disney often animated at 12 and rarely at 8, as well, and you can see it in any of their feature films, depending on the scene.

Creating Cel-like Limited Animations

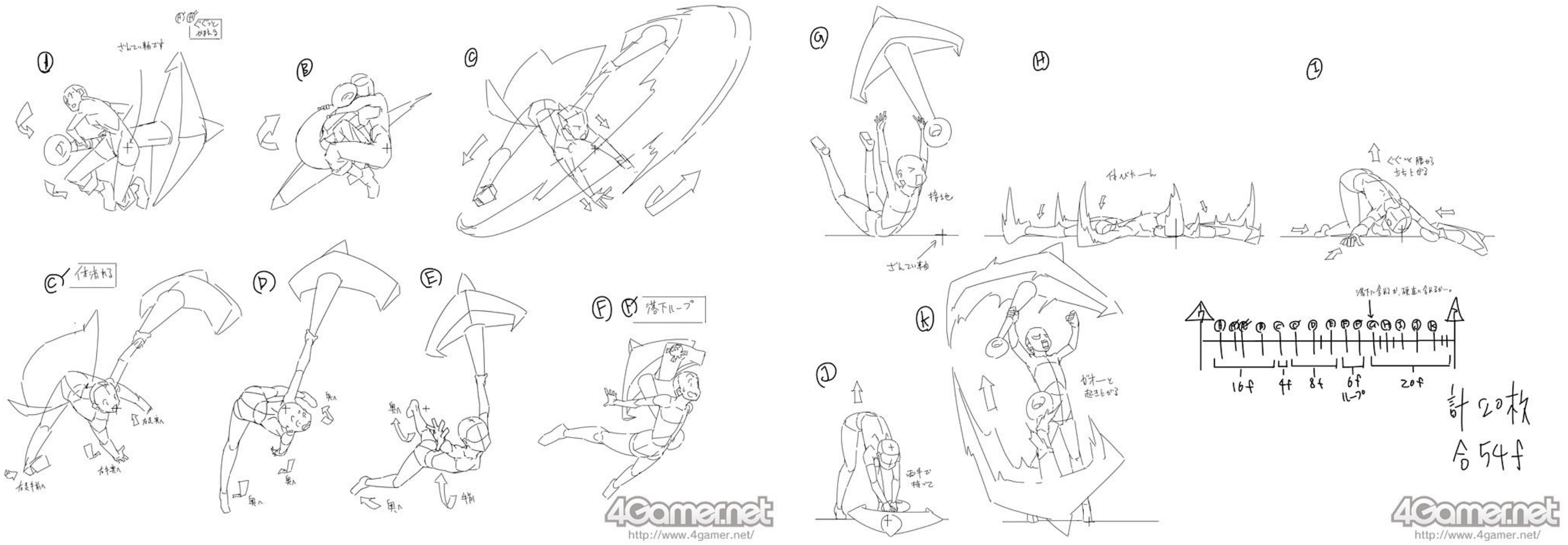
Example of the same animation in full frame rate and limited.

TN.: This video was uploaded before youtube supported full 60fps motion, so the example isn't as strong as with the final game.

Creating Cel-like Limited Animations

In the actual production process, a storyboard is created first. In this example, the specifications for a fighting game are set, such as for a special move: *"The entire move will be composed of 60 frames"* and *"At the 30th frame, a punch will be delivered, and the hitbox will appear at this timing"*. The storyboard is then given to the animator, who decides the pose of the character model for each frame.

The artists at Arc System Works are professionals in the creation of 2D fighting game action, so they must be familiar with such character poses. ***"We don't rely on interpolation using F-curves and such"*** Sakamura says.



Storyboard for May's Special Move.

Creating Cel-like Limited Animations

20 frames of the previous animation, show in 2 second intervals.

Creating Cel-like Limited Animations

What does it mean to say that the game displays at 60 frames per second even though it is a limited animation? It means "60 frames per second as a video specification".

For example, under 60fps display settings (16.67ms per frame) if a pose is designed to take 2F time, the same pose will be displayed for 33.33ms (= 16.67ms x 2).

Movements of parabolic trajectories that occur when characters run, jump or fly through the air are still updated at smooth 60 fps. Also, I don't think it needs mentioning, but the timing of the player's command input is accepted at intervals of 1/60th of a second.

D. Ishiwatari— *When posing a character in a cutscene, we adjust the position of each part of the character model and the position of the light source for that character, frame by frame, in order to improve the appearance of the cutscene. In the case of animations where the camera goes around the character, I also adjust the position of the nose for each frame (laughs).*

The unique graphical expression of GGXrd would not have been possible without not only the advanced technology, but also **the accumulation of these skilled animators' artistic senses**.

TN.: Here the author repeats a misconception about Disney and full animation. Disney often animated at 12 and rarely at 8, as well, and you can see it in any of their feature films, depending on the scene.

Creating Cel-like Limited Animations

Limited animation as showcased in May’s Instant Kill. No sound.

TN.: Video at half speed, 30fps. Play at double speed to see how it looks in-game.

Making the Fighting Feel 2D

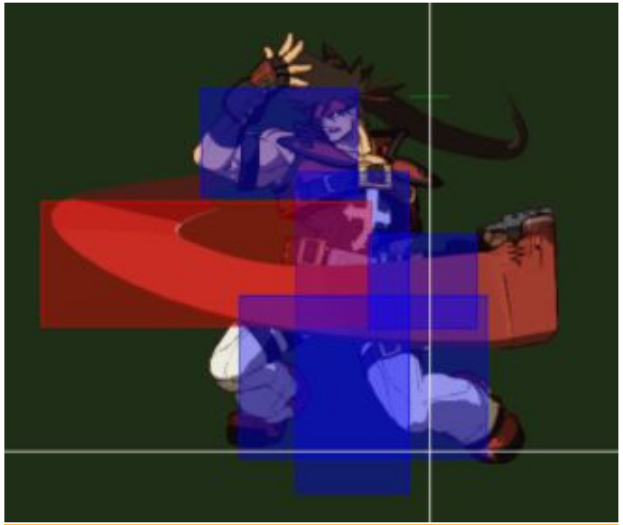
Setting up collision and mitigating 3D

Making the Fighting Feel 2D

The various efforts made to make the game look like a 2D fighting game while using 3D graphics paid off, and they were able to reach a level where the screenshots could be judged as "2D graphics" by the untrained eye.

However, the development team was still not satisfied and made further improvements to make the gameplay feel more like a 2D fighting game. Yamanaka looks back on this process:

T. Yamanaka— *In GGXrd, even though the graphics are 3D-based, **the gameplay is a 2D fighting game**, so we didn't want to use spheres, cubes or capsules like in a typical 3D game.*


This tool is called “Collision Setting Tool” at Arc System Works. Collisions are created as rectangles, same as 2D games.

The collision detection area, specifically the hurtbox (the area where damage is taken when an enemy attacks) and the hitbox (the area that inflicts the damage), were set using familiar in-house tools, as briefly mentioned in the previous part. Each frame of the character's motion created by the limited animation system was considered as 2D graphics, and the 2D collision detection area was set for each frame using in-house tools.

Making the Fighting Feel 2D

However, there was an issue. **3D graphics are the result of cutting out and displaying the view from a viewpoint on a rectangular screen**. However, in the first place, the field of view that we are looking at is one that is reflected on the inner wall of a sphere, whether we are aware of it or not. The problem is how to see the "spherical inner wall view" when it is projected onto a rectangle. In a racing game, you may have noticed that when a car approaches you from behind, into your field of vision, it may look strange and stretched out.

In fact, in GGXrd, when the characters were rendered without any changes, **they tended to look a little wide at the left and right edges of the screen, and a little thin in the center.** If this were left unchecked, the aforementioned "2D collision data" won’t correspond, depending to a character’s position on-screen..

To solve this problem, you can either adjust the collision detection according to the position of the character in the screen, or you can mitigate the "fatness" caused by the position of the character in the screen.

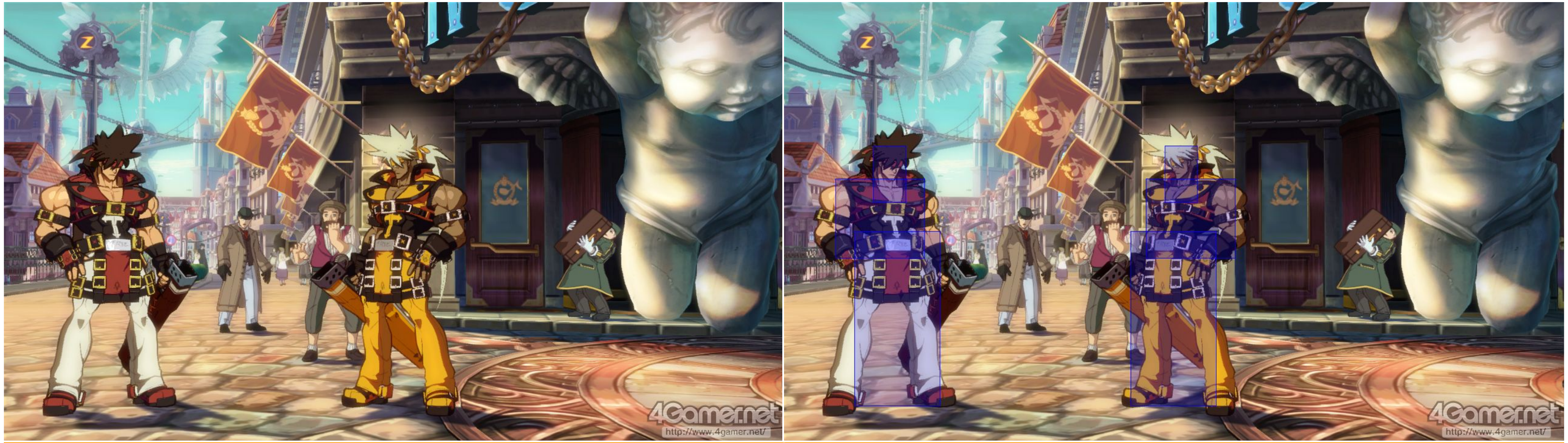
The development team chose the latter approach, which was a natural choice, since **in 2D fighting games the appearance of the characters should not change just because they are positioned in the left, right, or center of the screen.** Incidentally, Street Fighter IV series, the predecessors of 2D fighting games based on 3D graphics, took this same approach.

The actual adjustment was done by changing the method of drawing the 3D model on the 2D screen. There are two types of 3D to 2D projection: perspective projection, which makes near objects appear larger and far objects appear smaller, and parallel projection (orthographic projection), which makes no such difference.

This prevents the characters from getting wider or thinner depending on their screen position, resulting in an almost constant size at all positions.

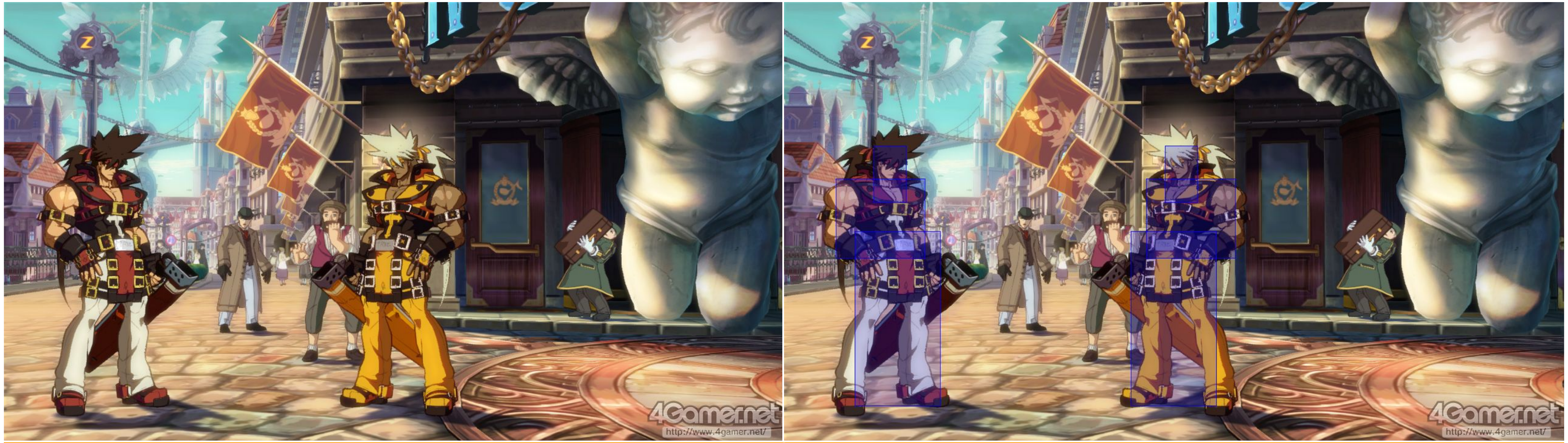
Making the Fighting Feel 2D

D. Ishiwatari— *Strictly speaking, the same kind of hybrid projection should be applied to the vertical direction as well, but it's not as critical as the horizontal direction, even if it's not applied, and the test players didn't feel any discomfort, so we decided not to introduce it.*



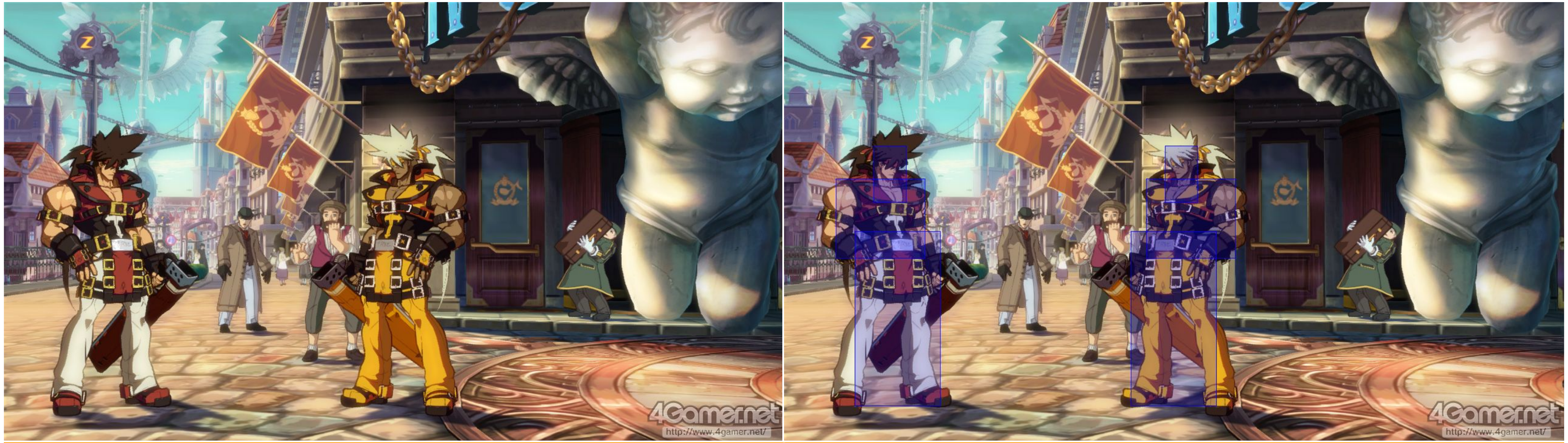
A screenshot with 100% perspective projection (left) and hitbox displayed.

Making the Fighting Feel 2D



A screenshot with 100% parallel projection and hitbox displayed.

Making the Fighting Feel 2D



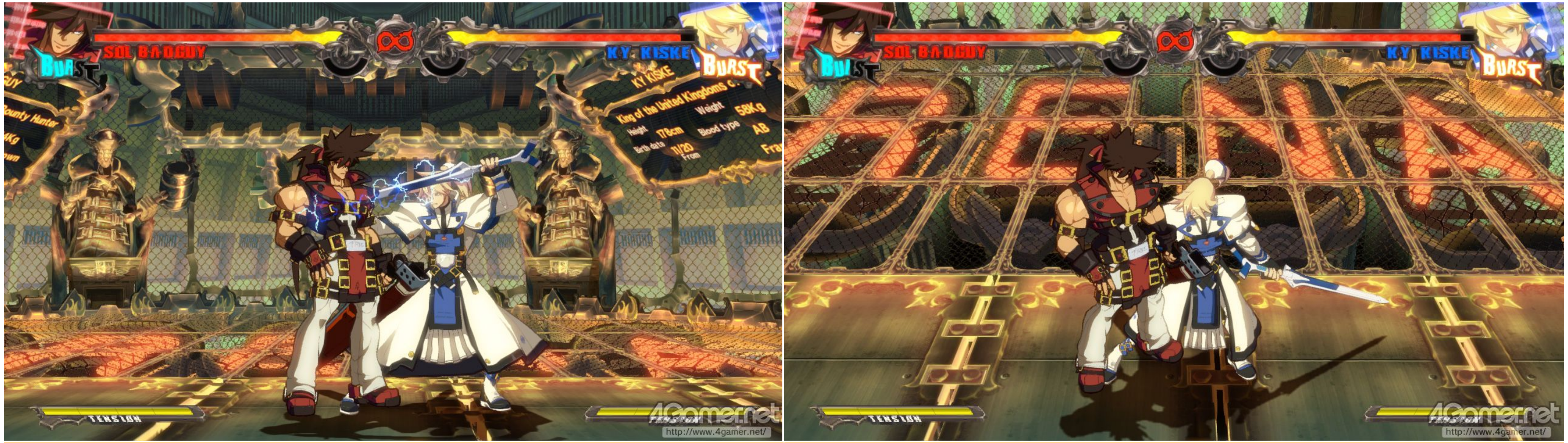
Final version, a combination of 30% perspective and 70% parallel projection (left) and hitbox displayed.

Making the Fighting Feel 2D

D. Ishiwatari— *Another thing that we had to introduce was a processing system for situations where two characters overlap.*

With pixel sprites, one of the characters is always above the other, but in 3D, each character has its own three-dimensional size, so when two characters are close together, one character's protruding arm may be clipped into the other character. We had to find a way to avoid this.

The final solution was to offset the depth (=Z) value of the attacker's character by about one meter in 3D space, so that the overlap would not occur. In short, the attacker's character is drawn so that it always passes the depth (=Z) test, ignoring the 3D back and forth.

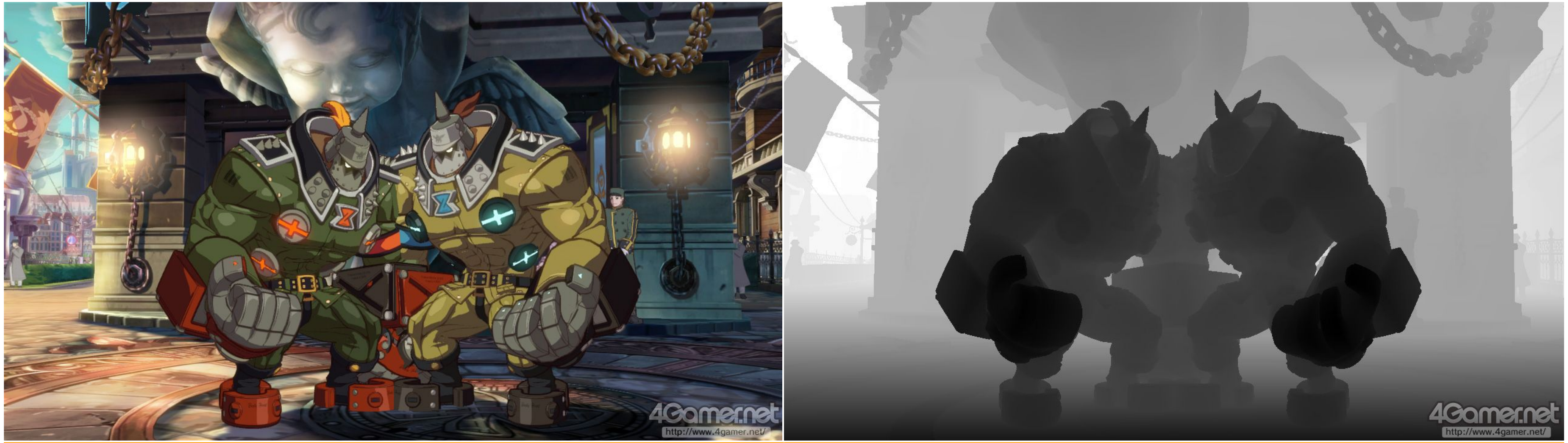


In the screenshot on the right, the camera is moved while Sol and Ky are facing each other (left). You can see that the characters are on the same axis.

Making the Fighting Feel 2D

J. Motomura— *The two characters are fighting each other, and in terms of 3D coordinate processing, they are on the same Z axis. So the only intervention is in the drawing process.*

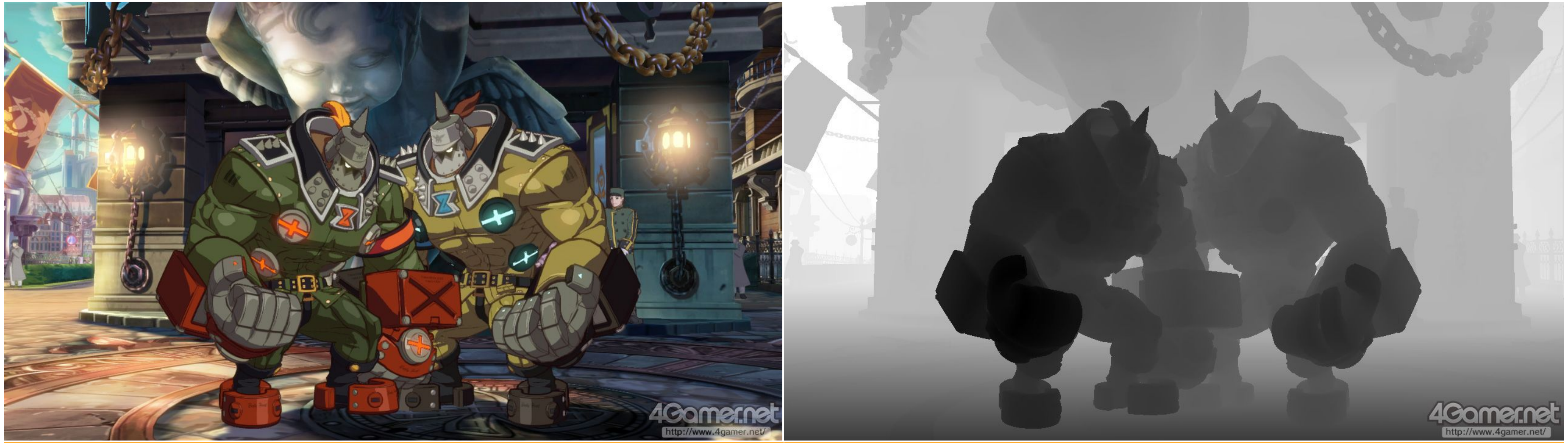
In the case of a move where the character grabs the opponent with both arms and throws him, the offset is adjusted so that the thrower is inside the thrower's arms. Some parts are covered up by the flame effects (laughs).



Character rendering before depth adjustment (left) and its Z-buffer (right), where the two characters overlap in the center.

Making the Fighting Feel 2D

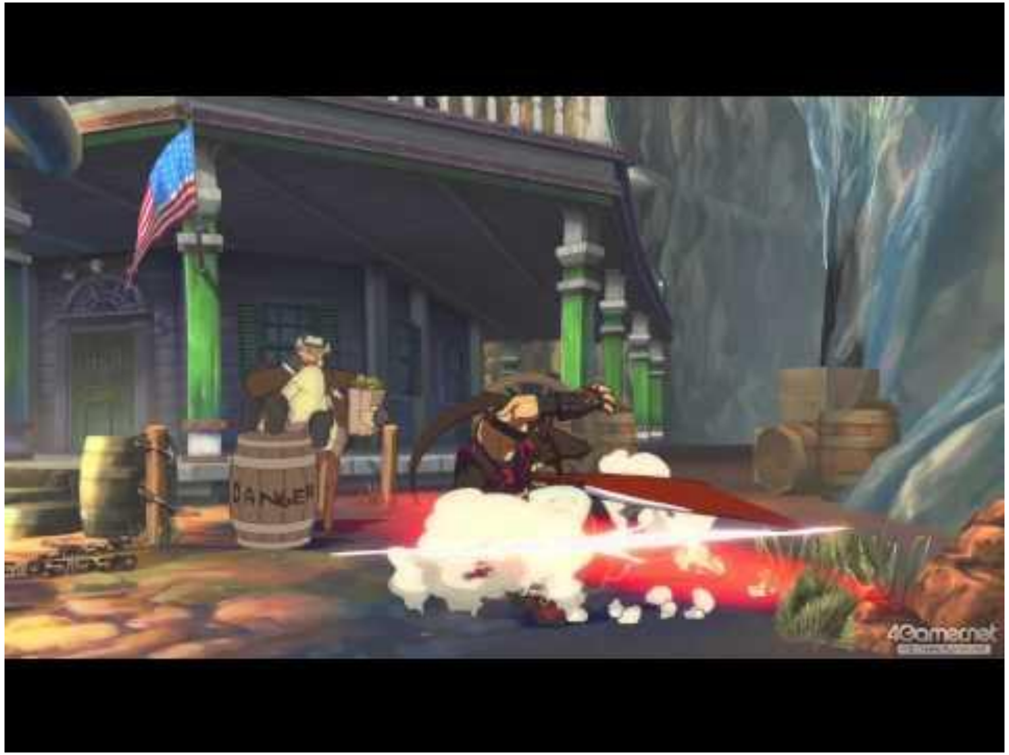
With the implementation of this system, most situations are no longer unnatural, but there are some cases where characters that are large in the depth direction, such as the giant Potemkin character, are in close contact with each other. However, it was judged that it would not interfere with play, so it was left as is.



Character rendering after depth adjustment (left) and its Z-buffer (right)

Making the Fighting Feel 2D

Rotating the camera during a battle scene in GGXrd.



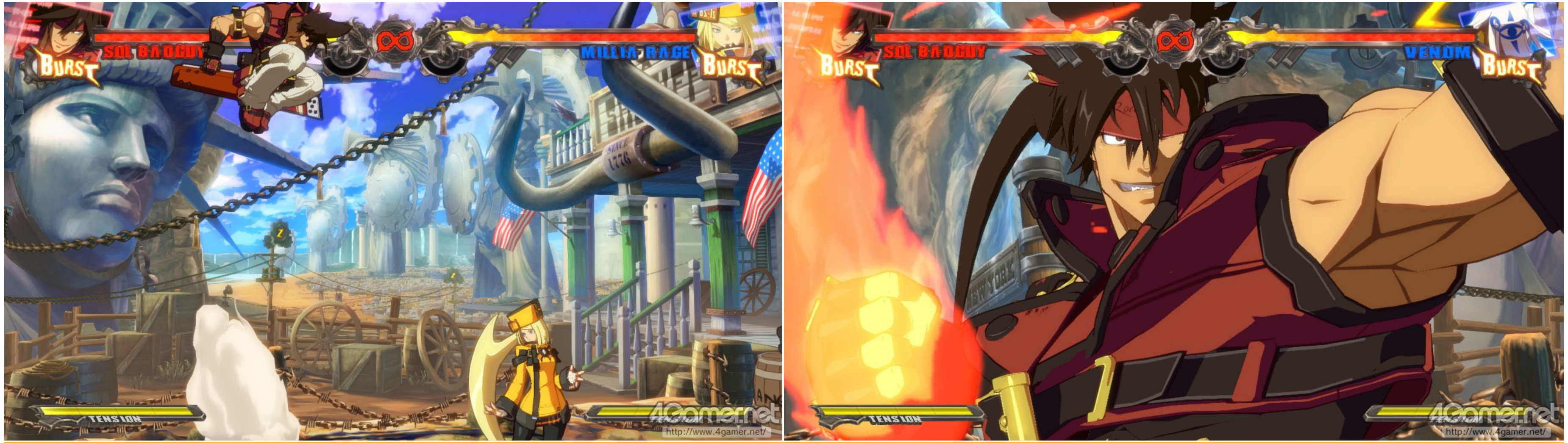
Making the Fighting Feel 2D

Rendering the gauges and flipping characters

Making the Fighting Feel 2D

The screen is designed to be played as a 2D fighting game, and this is also reflected in the rendering of various gauges such as health. Textures are rendered onto planar polygons, which are placed in a Z position behind the fighting character, so when a character jumps high and overlaps the gauge, the gauge will be drawn behind them. There are exceptions to this, such an attack that launches you into the air, where the gauge is drawn in front.

J Motomura— *When the camera angle changes drastically, we foresaw that the gauges would interfere with the background, so we disabled the special processing that placed the gauges behind the character and placed them in front. This was based on the idea that the gameplay itself is temporarily suspended during this camera angle effect, so placing the gauges in the front would not cause any stress to the player.*



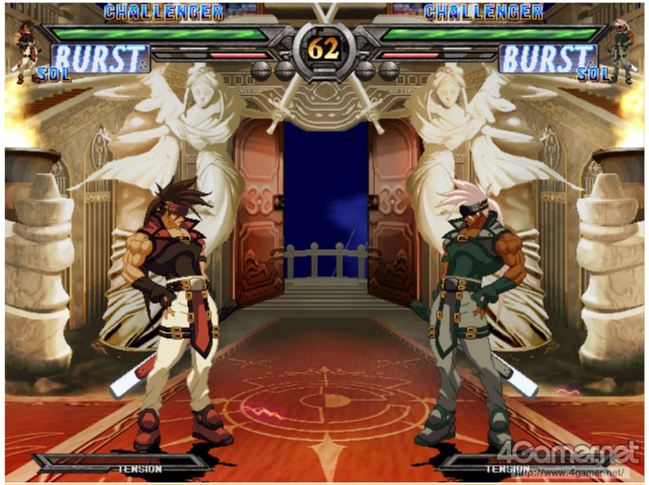
Gauges displayed behind the character (left) in normal battle scenes, and in front (right) when the camera angle changes.

Making the Fighting Feel 2D

In the real world, when two fighters face each other in a fighting pose with a shoulder forward, one fighter will have his chest facing the camera and the other will have his back facing the camera. But in a 2D fighting game, the on-screen "appearance" of the character changing depending on whether the character is facing left or right, would make it difficult to play.

Traditionally, games simply flipped the sprites horizontally. So, can we do the same thing in 3D for GGXrd?

Actually, if you just want to flip a character model symmetrically, all you have to do is flip the 3D model with respect to the vertical axis (Y-axis) (i.e., swap the positive and negative X-coordinate values of the vertices that make up the 3D model) and draw it. Of course, in order to flip the lighting results as well, the aforementioned character's exclusive light source will need to be flipped to match the 3D model.

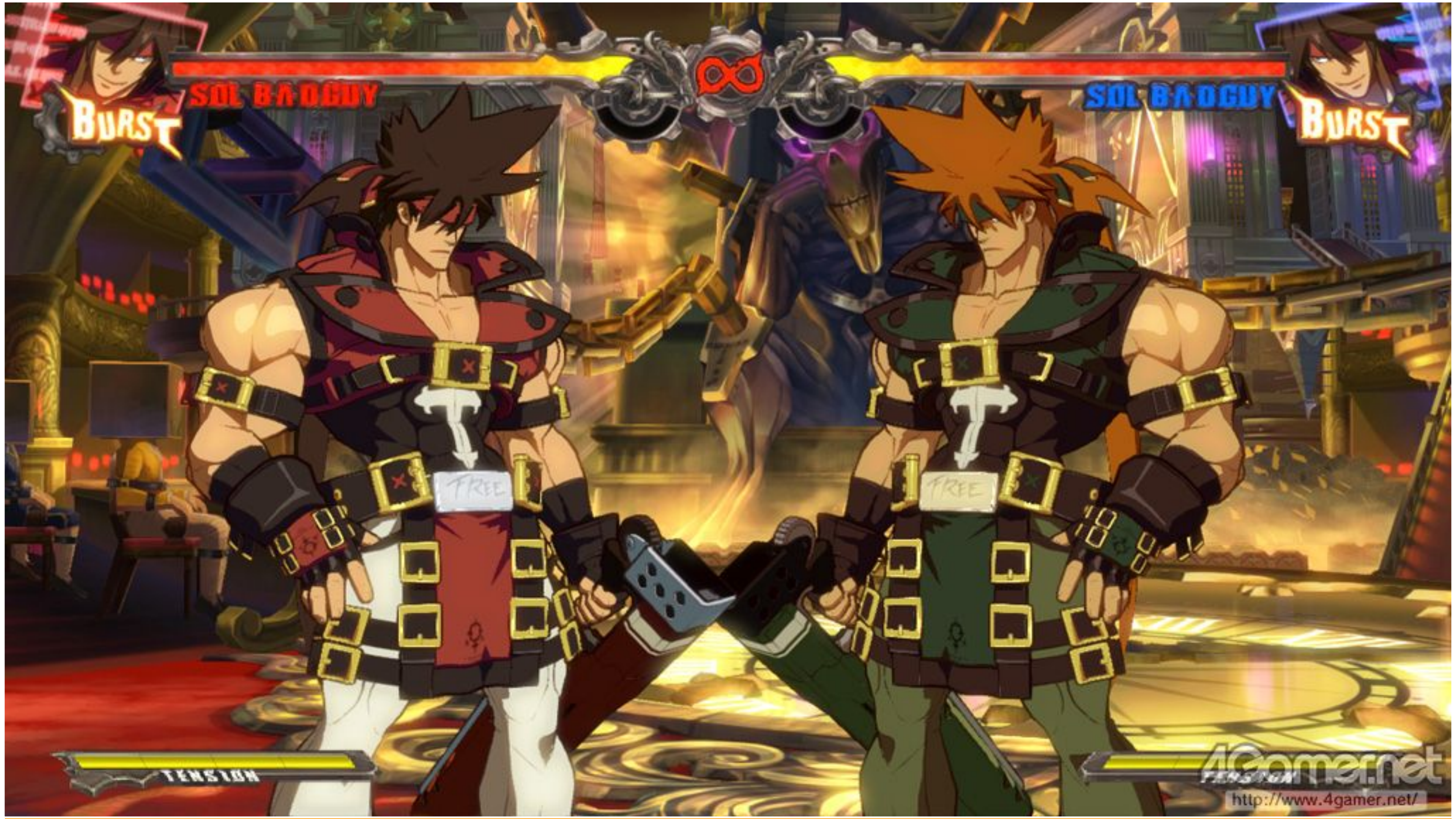


A battle screen in the 2D prequel
GUILTY GEAR XX #RELOAD © ARC SYSTEM WORKS

A potential problem is the text on some of the character costumes. For example, Sol's belt has the word "FREE" engraved on it, but if you simply flip the 3D model of the character, the text will be mirrored.

The game avoids this problem by flipping the UV map for the area where the text is drawn so that the text is always texture mapped in the forward direction when the 3D model is flipped.

Making the Fighting Feel 2D



A closer look at the characters during battle. Not the “FREE” on the belt and the decals on the gloves.

Making the Fighting Feel 2D

J. Motomura— *For texturing, we have implemented another little trick: we have doubled the luminance of the texture before combining it with the base texture. Since the designed texture luminance is stored multiplied by 0.5, when it is doubled, it becomes 1.0 and the texture is applied as designed. If the brightness of the texture image exceeds 0.5, it will be brighter than the ground color. By using this system, you can easily create a "three-dimensional design with shaded text" kind of look.*

The decals are attached to polygons, like a translucent sticker, over the surface of a 3D model, similarly to a plastic model kit. The system was originally designed to ensure sufficient resolution by separating the base texture from text.

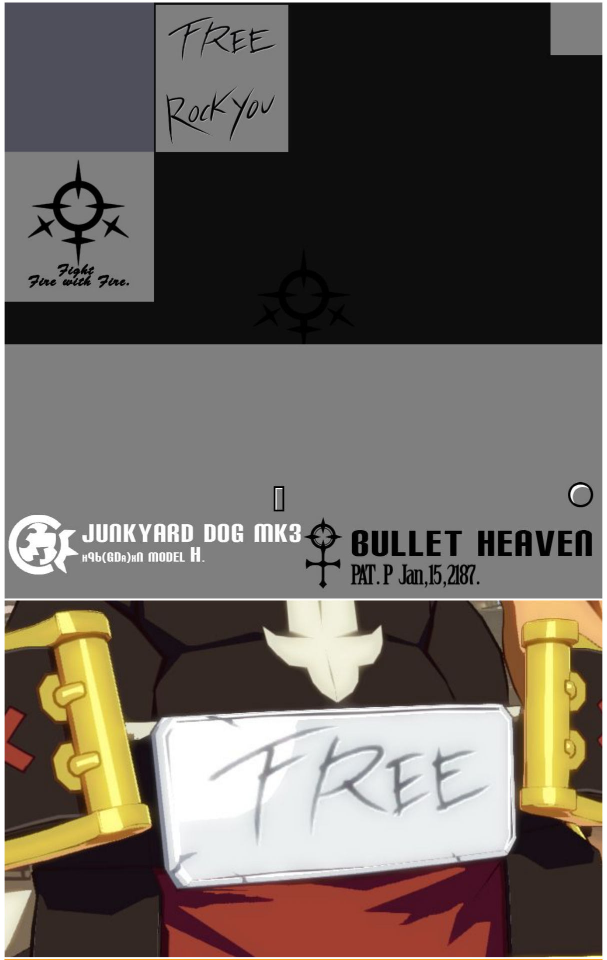
When rendering these decals, it is important to know how to blend them with the base texture.

Assuming that "Dest" is the base texture and "Source" is the character texture, the formula is as follows.

Dest x Source + Source x Dest = twice the result of the multiplication

In this case, the texture content is multiplied by a factor of two to the base texture, so the result is "darker when less than 0.5, transparent when 0.5, and lighter when over 0.5. This blending method was not a standard feature in UE3, so Ieyumi added it.

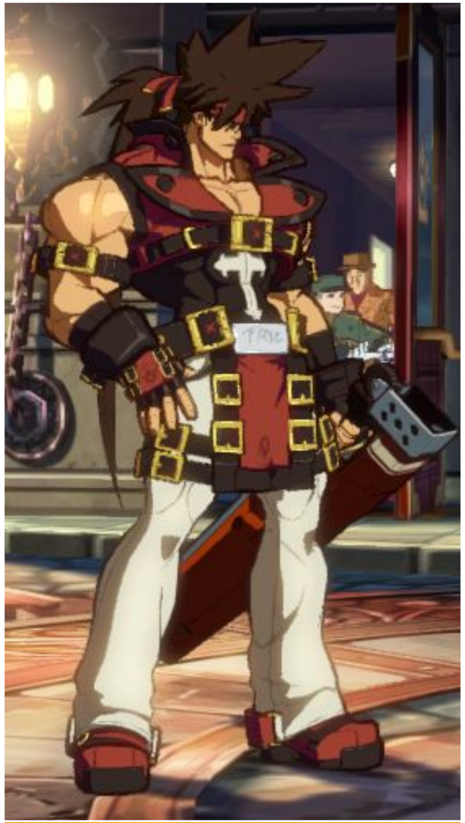
If you try to do shaded text in a simpler way, you would have to texture map the textures for the light and shadow areas of the text. This technique, however, can be achieved with a single pass of texturing.



Sol’s texture atlas and belt. The base color is a neutral gray.

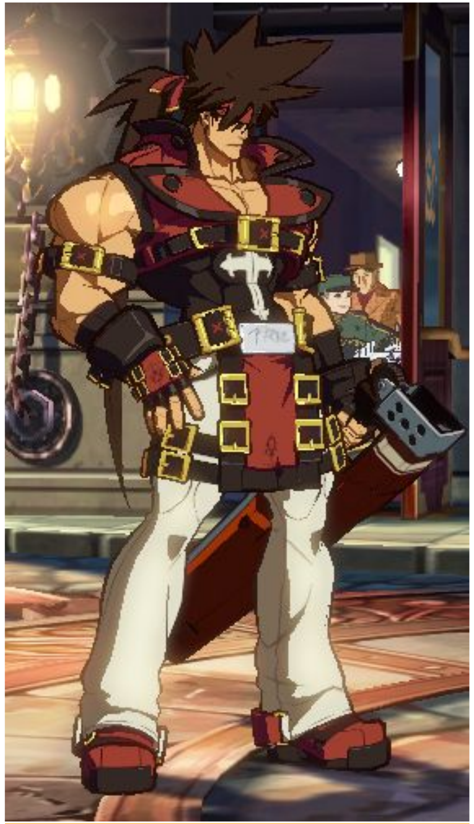
Effective Use of Post-Processing

Effective Use of Post-Processing

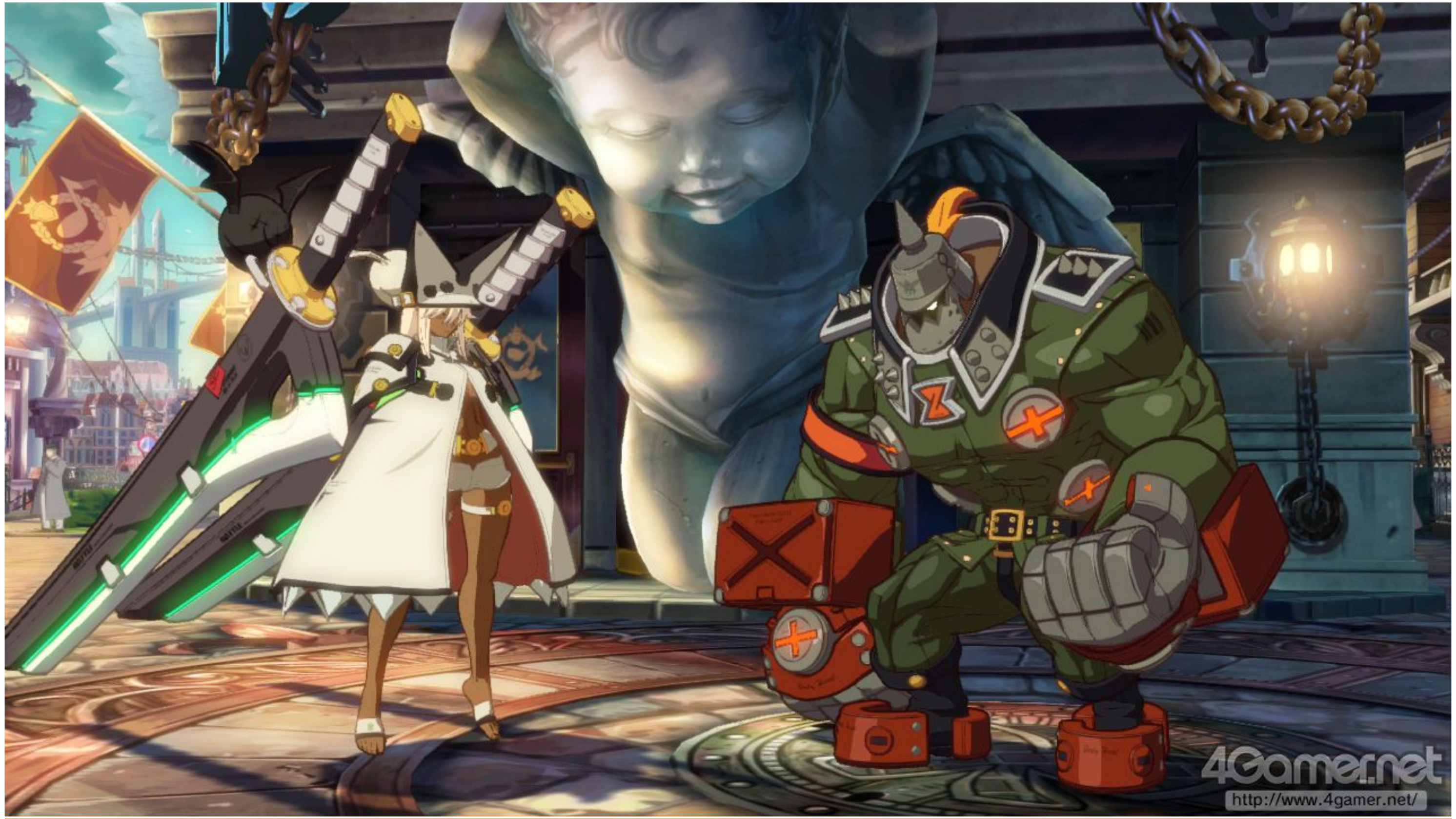

With FXAA

In recent years, photo-realistic 3D game graphics have tended to include a wide variety of post-effects. In contrast, GGXrd, with its cel-shaded animation style as its goal, is relatively simple.

As mentioned before, the game uses FXAA as post-processing anti-aliasing. It uses a 64-bit buffer with 16-bit floating-point alpha and 16-bit floating-point RGB as render targets, and implements a pipeline that supports HDR (High Dynamic Range) rendering, so high-luminance areas overflow. A light bloom post-effect is also used.

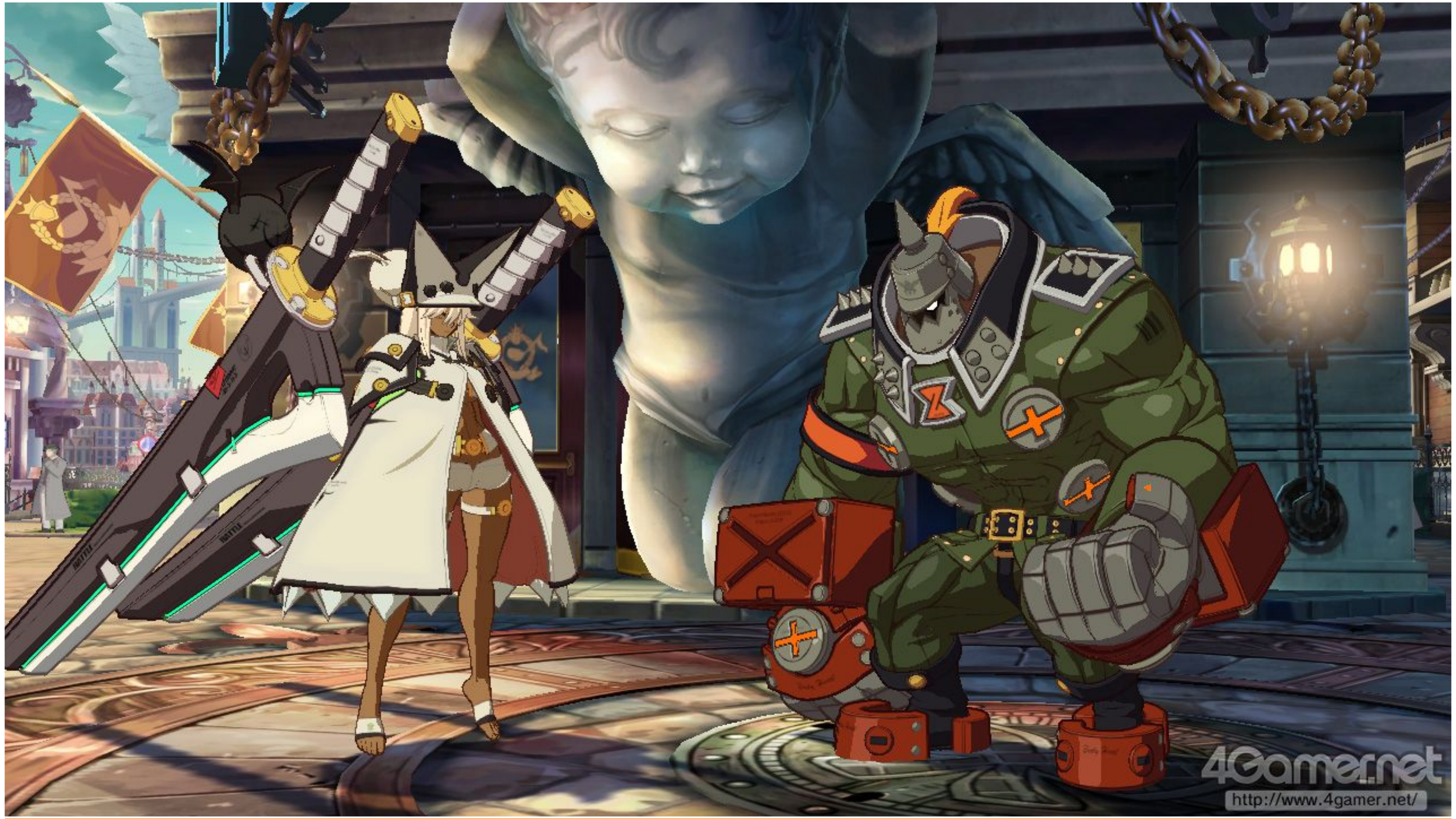

Without FXAA

Effective Use of Post-Processing



The game with all post-processing active.

Effective Use of Post-Processing

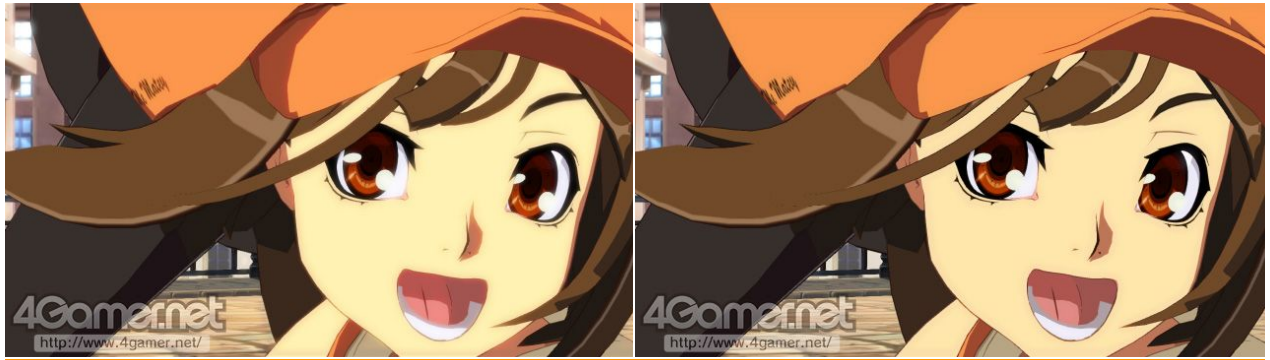


The game with post-processing disabled.

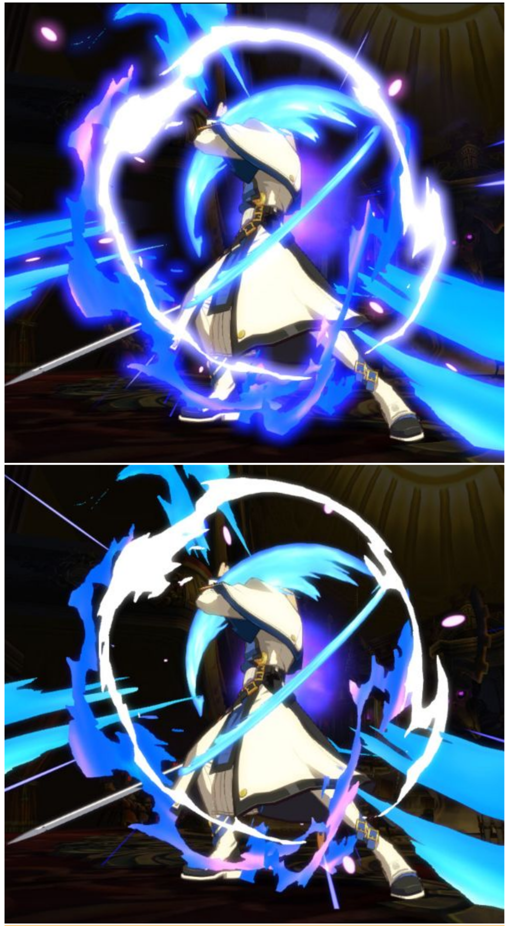
Effective Use of Post-Processing

D. Ishiwatari— *This kind of light bloom effect is commonly used in recent anime, so I don't think it feels out of place. In addition, we also use post-effects to create the Diffusion effect.*

J. Motomura— *Yes. Diffusion is useful for adding a softer touch to an animated image that is divided into two parts, light and dark. While light bloom is applied to brightness values of 1.0 and above, diffusion is applied to lower brightness values.*

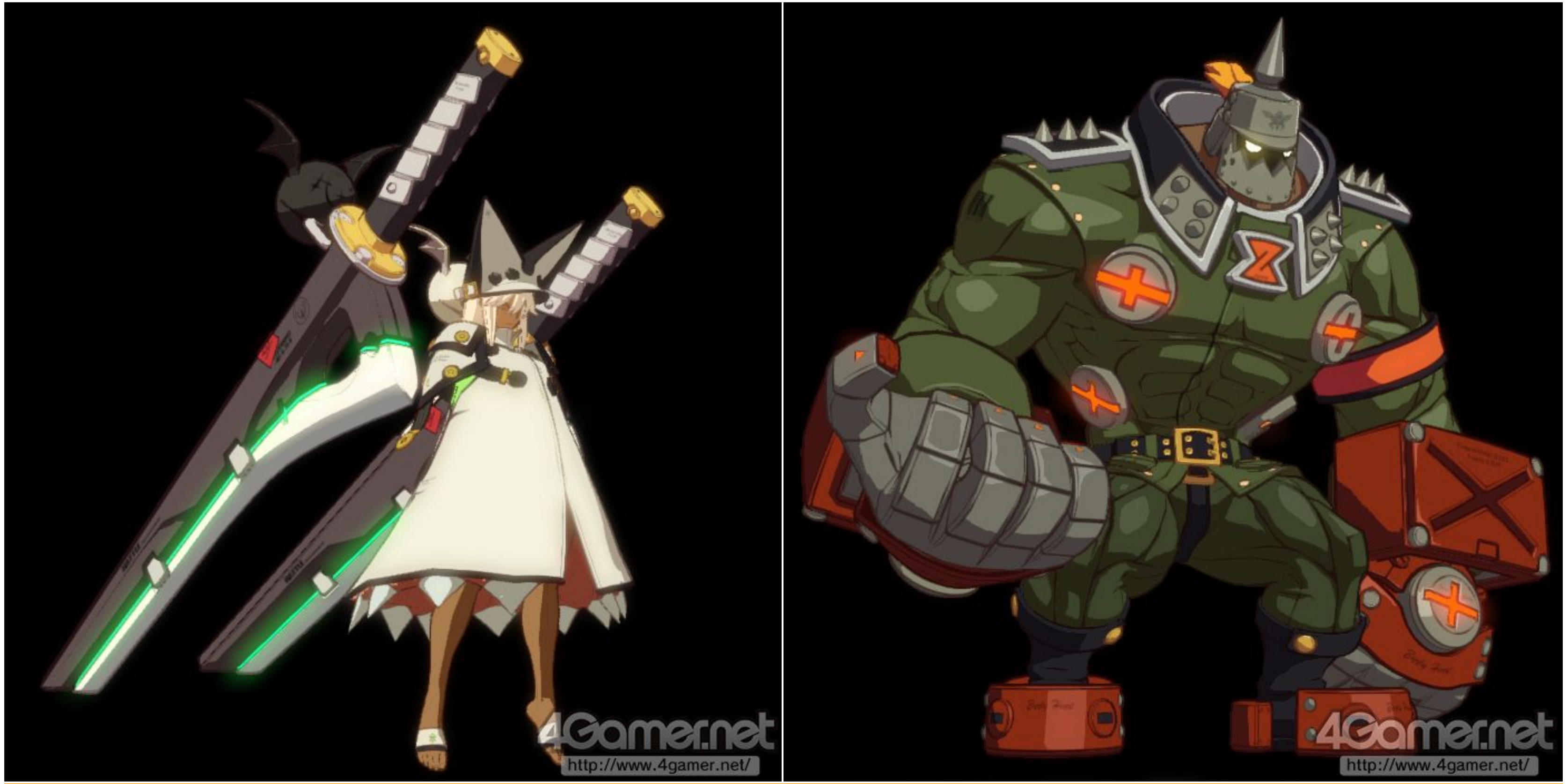


May with(top) and without (bottom) diffusion.



With(top) and without (bottom) light bloom.

Effective Use of Post-Processing



Ramlethal and Potemkin with their luminescent textures.

Final Word

Final Word

Looking at the final game screen, it looks so natural as 2D graphics that most players wouldn't even recognize its entirely 3D.

At first the development team was concerned about whether the 3D graphics system would be able to reproduce 2D fighting game graphics, but by the time development was complete, they were more concerned about whether people would notice that it was even 3D at all. They wanted to incorporate more "3D aspects" into the game system, but for an arcade game, they decided to prioritize **ease of play and a sense of comfort as a 2D fighting game** and settled on the current state.

Hopefully, the PS4 and PS3 versions, will include such extra elements that are "only possible with 3D graphics".

The accumulated a lot of know-how in "2D visuals using 3D graphics" through this GGXrd project. At the same time, they feels that there are still unexplored areas of this method, and they would like to continue to develop technology in this vein for use in future titles.

While it is true that there are fields such as "NPR" (Non Photorealistic) visuals and "Stylized Rendering" in 3D graphics, there are very few that specialize in the Japanese anime style. In order to develop in this field, it is necessary to have the ability and knowledge to theoretically analyze the style of Japanese animation, and in this sense, the Arc System Works development team is well qualified. I hope they will surprise us again by not only developing new technology, but also improving their ability to apply it, for example by applying it to games other than 2D fighting games.

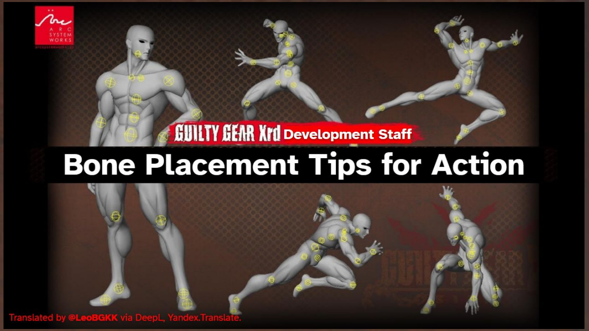
Other GGXrd 3D Materials

GDC THE ART STYLE OF GUILTYGEARXRD

GUILTY GEAR Xrd Development Staff

Anime Style Character Modeling TIPS

Translated by @LeoBGKK via DeepL, Yandex.Translate.


GUILTY GEAR Xrd Development Staff

Bone Placement Tips for Action

Translated by @LeoBGKK via DeepL, Yandex.Translate.

GUILTY GEAR Xrd Development Staff

Modeling for Skinning

Translated by @LeoBGKK via DeepL, Yandex.Translate.

The secrets behind
Guilty Gear Xrd -SIGN-'s real-time 3D Anime graphics

4Gamer.net

This new series, ***"Examining Game Graphics"*** focuses on the graphics of specific titles, and explains the systems and technology behind them. I have been developing ***"Nishikawa Zenji's 3D Game Ecstasy"*** as a series, but since the scope of coverage has become too broad, I would like to deal with technical explanations of specific titles in this new series.
This 1st commemorative issue will feature **GUILTY GEAR Xrd -SIGN-**, a fighting game developed by Arc System Works and rocking arcades since 2014/2.

Original Article .: Writer_西川善司/Zenji Nishikawa ; Photographer_佐々木秀二/Shuji Sasaki , 2014

Translation and slides by .: @LeoBGKK on Twitter
Additional translation by Chev on Polycount Boards

CGWORLD

Non-photorealistic **GUILTY GEAR Xrd -REVELATOR-** has evolved even further by adding features unique to 3D.

This series has been fascinating players with its 2D anime-like visuals and it's even more impressive with its latest instalment, GUILTY GEAR Xrd -REVELATOR-. This time, let's focus on elements other than the ones we've previously covered (CGWORLD vol.214)

ORIGINAL ARTICLE: TEXT_武田かおり/ KAORI TAKEDA, CGWORLD Vol.215
EDIT__藤井紀明 / Noriaki Fujii (CGWORLD)、山田桃子 / Momoko Yamada

Translation and slides by @LeoBGKK on Twitter

Follow me on twitter.

Leo Bruno "GKK"

@LeoBgkk

🇵🇹 Leonardo Bruno. Garuda. Kanta. DM for COMISSION Info.
✏️Illustration // 📼Video // 🕹️ButtonMashing.

If you appreciate this translation and wish to donate or something, here's my paypal.
Other presentations will be linked here, as I finish translating them.